BlackBerry Java Application Core

版本：5.0

开发指南



发布日期： 2010-06-30
SWD-790641-0630011532-011

内容

管理应用程序

1

应用程序管理器

BlackBerry® 设备上的 BlackBerry® Java® Virtual Machine 包括应用程序管理器。这是其它 BlackBerry 设备应用程序的操作系统事件的中央分派程序。

net.rim.device.api.system.ApplicationManager 类让应用程序可与应用程序管理器交互，以执行以下操作：

- 与进程交互，如检索前台应用程序的 ID
- 将全局事件发布至系统
- 立即或在特定时间运行应用程序

检索有关 BlackBerry Java Application 的信息

1. 导入所需的类和接口。

```
import net.rim.device.api.system.ApplicationManager;
import net.rim.device.api.system.ApplicationDescriptor;
import java.lang.String;
import net.rim.device.api.system.*;
```

2. 要检索有关正在运行的进程的信息，请调用 ApplicationManager.getVisibleApplications()

```
ApplicationManager manager = ApplicationManager.getApplicationManager();
ApplicationDescriptor descriptors[] = manager.getVisibleApplications();
```

3. 要检索正在运行的 BlackBerry® 设备应用程序的对象的说明，请调用 ApplicationDescriptor.getName()。

```
String appname1 = descriptors[0].getName();
```

4. 要检索当前应用程序的说明，请调用 ApplicationDescriptor.currentApplicationDescriptor()。

```
ApplicationDescriptor descriptor =
ApplicationDescriptor.currentApplicationDescriptor();
```

与其它 BlackBerry Java Application 通信

1. 导入 net.rim.device.api.system.ApplicationManager 类。
2. 要将系统级事件发布至其它 BlackBerry® 设备应用程序，请调用其中一个 ApplicationManager.postGlobalEvent() 方法。

确定可用于 BlackBerry Java Application 的服务

服务预订包含服务记录。 每条服务记录都定义 BlackBerry® 设备上的某项服务。 服务记录定义通信协议（WAP 或 IPPP）、网络网关和配置信息，如浏览器设置。

1. 导入 net.rim.device.api.servicebook 类。
2. 要让 BlackBerry 设备应用程序与 BlackBerry® Infrastructure 交互，请使用服务预订 API (net.rim.device.api.servicebook)。

应用程序控制

BlackBerry® Application 控制 IT 策略规则将在应用程序在特定 BlackBerry 设备上运行时，让管理员可以建立应用程序的功能。 例如，管理员可使用 BlackBerry Application 控制 IT 策略规则确保 BlackBerry 设备上的游戏无法访问电话 API。 只有 BlackBerry 设备连接至 BlackBerry® Enterprise Server，BlackBerry Application 控制 IT 策略规则才能发挥作用。 此 IT 策略不适用于仅使用 BlackBerry® Internet Service 的 BlackBerry 设备。

如果管理员或用户拒绝应用程序访问受保护区域之一，则关联方法将引发 ControlledAccessException。对于类级检查，此方法将引发 NoClassDefFoundError。 您的应用程序可能需要处理这两种类型的错误，具体取决于使用的 API。

请求访问资源

1. 导入所需的类和接口。

```
import net.rim.device.api.applicationcontrol.ApplicationPermissions;
import net.rim.device.api.applicationcontrol.ApplicationPermissionsManager;
```

2. 创建 ApplicationPermissions 类的实例。

```
ApplicationPermissions permissions = new ApplicationPermissions();
```

3. 指定构建请求，以请求事件插入权限。

```
permissions.addPermission( ApplicationPermissions.PERMISSION_EVENT_INJECTOR );
```

4. 确定 BlackBerry® 设备用户指定的访问控制设置。

```
if(ApplicationPermissionsManager.getInstance().invokePermissionsRequest
(permissions))
{
   System.out.println("The user allowed requested permissions.");
}
else
```

```
{
   System.out.println("The user denied requested permissions.");
}
```

代码模块

检索模块信息

1. 导入 net.rim.device.api.system.CodeModuleManager 类。
2. 要检索模块的句柄，请调用 getModuleHandle()，并将代码模块的名称作为参数提供。

```
int handle = CodeModuleManager.getModuleHandle("test_module");
```

3. 要检索有关模块的特定信息，请调用 CodeModuleManager 类的方法，并将模块句柄作为参数提供给这些方法。

```
String name = CodeModuleManager.getModuleName( handle );
String vendor = CodeModuleManager.getModuleVendor( handle );
String description = CodeModuleManager.getModuleDescription( handle );
int version = CodeModuleManager.getModuleVersion( handle );
int size = CodeModuleManager.getModuleCodeSize( handle );
int timestamp = CodeModuleManager.getModuleTimestamp( handle );
```

4. 要在 BlackBerry® 设备上检索现有模块的句柄数组，请调用 getModuleHandles()。

```
int handles[] = CodeModuleManager.getModuleHandles();
String name = CodeModuleManager.getModuleName( handles[0]);
```

访问控制消息

为需要用户许可的操作显示消息

您可以使用 net.rim.device.api.applicationcontrol 数据包的组件，以在应用程序尝试必须获得用户许可的操作时，让 BlackBerry® 设备应用程序可向 BlackBerry 设备用户显示自定义消息。 应用程序显示有关用户必须提供的许可类型的信息。 例如，可将 PERMISSION_PHONE 用于需要访问设备的电话功能的操作。

您可以使用 applicationcontrol 数据包包括自定义消息，以及应用程序响应应用程序控制将显示的默认消息。

向用户显示应用程序控制消息

BlackBerry® 设备应用程序可包括多个注册的 ReasonProvider。应用程序将按每个 ReasonProvider 在应用程序中的注册顺序显示来自 ReasonProvider 的消息。 例如，如果应用程序将 ReasonProvider A 注册在 ReasonProvider B 之前，则应用程序将先显示来自 ReasonProvider A 的消息，然后再显示来自 ReasonProvider B 的消息。

1. 导入 net.rim.device.api. applicationcontrol.ReasonProvider 接口。
2. 在 ReasonProvider.getMessage(intpermissionID) 方法的实施中,返回包含要向用户显示的消息的 String 值。

存储数据

2

数据管理

您可将数据存储至 BlackBerry® 设备上的持久内存。 BlackBerry 持久存储区 API 和 MIDP RMS API 提供 JSR 37 和 JSR 118 支持。 运行 BlackBerry® Device Software 4.2 版或更高版本的 BlackBerry 设备还提供传统文件系统和支持，用于通过 JSR 75 API 直接将内容保存至文件系统。 通过 BlackBerry 持久存储区 API 或 MIDP RMS API，可将数据持久存储至闪存。 即使从 BlackBerry 设备卸下电池，数据也将存留。

运行 BlackBerry® Device Software 5.0 版或更高版本的 BlackBerry 设备支持使用 SQLite®。 您必须使用 JSR 75 API 或 SQLite，才能将数据存储在媒体卡上。

内存访问

BlackBerry® Java® 环境 旨在阻止应用程序无意或恶意在其它应用程序或 BlackBerry 设备上引发问题。BlackBerry 设备应用程序仅可写入 BlackBerry Java Virtual Machine 使用的 BlackBerry 设备内存；它们无法访问其它应用程序的虚拟内存或持久存储区（除非特别为它们授予此类访问权限）。 自定义应用程序只能通过特定 API 访问持久存储区或用户数据，或与其它应用程序通信。 Research In Motion 必须以数字方式对使用某些 BlackBerry API 的应用程序进行签名，以提供使用敏感 API 的应用程序的审核跟踪。

文件系统和路径

BlackBerry® 设备上的文件系统具有用于访问它们的对应文件路径。

文件系统安装点	路径
内部闪存	file:///store/。
媒体卡	file:///SDCard。

文件连接 API

文件连接 API 提供传统文件系统，且支持将数据直接保存至 BlackBerry® 设备上的文件系统或保存至 microSD 卡。 您可以在文件系统中查看数据，以及使用 Windows® 将数据移至计算机。

存储在可移动介质上

访问 microSD 媒体卡上的数据

javax.microedition.io.file数据包支持 JSR 75 文件连接 API，且在应用程序中用于访问 microSD 媒体卡的文件系统。 您也可以实施 FileConnection 接口，以访问 BlackBerry®设备铃声和相机图像。

类或接口	说明
ConnectionClosedException	应用程序在关闭的文件连接上调用方法时，将引发此例外。
FileConnection	应用程序可使用此 API 访问文件或目录。
FileSystemListener	应用程序添加或删除文件系统根时，可使用此 API 接收状态通知。
FileSystemRegistry	应用程序可将此 API 用作监听文件系统的添加或删除的文件系统监听器的中心注册表。
IllegalModeException	方法需要特定安全模式（如 READ 或 WRITE）但打开的连接未处于该模式时，将引发此例外。

将 microSD 媒体卡用于多部 BlackBerry 设备

如果 BlackBerry® 设备用户将 microSD 媒体卡移至未使用设备密码或使用的密码未成功解密 microSD 媒体卡主密钥的 BlackBerry 设备，BlackBerry 设备将提示 BlackBerry 设备用户输入 microSD 媒体卡密码。 如果 BlackBerry 设备具有密码，BlackBerry 设备用户可使用提示将 microSD 媒体卡密码更改为 BlackBerry 设备密码。

microSD 卡上的文件加密

IT 策略和 microSD 媒体卡

您可以将写入 microSD 媒体卡的 IT 策略加密数据应用于任何存储在 microSD 媒体卡上的新文件或修改过的文件。 此处仅加密管理员设置 IT 策略后才存储在 microSD 媒体卡上的文件。 除了多媒体文件，所有内容均加密。

microSD 媒体卡上的数据加密

BlackBerry® Java® Application 访问 microSD 内存卡上的文件时，将发生文件解密，文件也将移至应用程序将读取的主内存。 要让 BlackBerry Java Application 访问受密码保护的文件，BlackBerry 设备不能锁定。 已加密文件具有 .rem 扩展名，且无法在 BlackBerry 之外的平台上解密。

如果 NVRAM 已删除且使用 BlackBerry 设备密钥锁定 microSD 媒体卡，则不再可访问 microSD 媒体卡上的数据。 要删除不可访问的数据，请启动 BlackBerry 设备并删除所有已加密多媒体文件。

BlackBerry 设备使用存储在 microSD 媒体卡上的主密钥加密 BlackBerry 设备多媒体文件。 主密钥可防止 BlackBerry 设备必须在禁用加密或更改密码时加密或重新加密所有多媒体文件。

代码示例：读取二进制文件的各个部分

此代码示例演示如何通过从 .gif 文件读取题头信息，读取二进制文件的各个部分。 应用程序将从题头读取图像的宽度和高度。 要运行此代码示例，必须在 BlackBerry® 设备中将 .gif 文件放在媒体卡的根文件夹中。

```
import net.rim.device.api.ui.*;
import net.rim.device.api.io.*;
import javax.microedition.io.file.*;
import javax.microedition.io.*;
import java.io.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
public class RandomFileAccess extends UiApplication
{
    public static void main(String[] args)
    {
       RandomFileAccess app = new RandomFileAccess();
       app.enterEventDispatcher();
    }
    public RandomFileAccess()
    {
       pushScreen(new HomeScreen());
    }
}
class HomeScreen extends MainScreen
{
    public HomeScreen()
    {
          setTitle("Random File Access Sample");
          try
          {
             FileConnection fc = (FileConnection)Connector.open("file:///SDCard/
test.gif");
```

```
            boolean bFileExists = fc.exists();
            if (!bFileExists)
            {
              Dialog.alert("Cannot find specified GIF file.");
              System.exit(0);
            }
            DataInputStream in = fc.openDataInputStream();
            byte[] widthBytes = new byte[2];
            byte[] heightBytes = new byte[2];
            if ( in instanceof Seekable )
            {
               ((Seekable) in).setPosition(6);
               in.read(widthBytes,0,2);
               ((Seekable) in).setPosition(8);
               in.read(heightBytes,0,2);
            }
            int widthPixels  = widthBytes[0]  + 256 * widthBytes[1];
            int heightPixels = heightBytes[0] + 256 * heightBytes[1];
           add(new LabelField("Width: " + widthPixels + "\nHeight: " + heightPixels));
            in.close();
            fc.close();
        }
        catch (IOException ioe)
        {
            ioe.printStackTrace();
        }
    }
}
```

持久数据存储

创建持久数据存储区

每个 PersistentObject 都具有唯一长密钥。

1. 导入以下类：
   - net.rim.device.api.system.PersistentObject
   - net.rim.device.api.system.PersistentStore
   - java.lang.String
   - net.rim.device.api.ui.component.Dialog
2. 要创建唯一长密钥，请在 BlackBerry® Integrated Development Environment 中键入字符串值。
   ```
   com.rim.samples.docs.userinfo
   ```
3. 右键单击该字符串并单击**将 com.rim.samples.docs.userinfo 转换为长**。

4. 在代码中包括注释，以指明用于生成唯一长密钥的字符串。
5. 要创建持久数据存储区，请创建单一静态 PersistentObject 并调用 PersistentStore.getPersistentObject，其中将唯一长密钥用作参数。

```
static PersistentObject store;
static {
store = PersistentStore.getPersistentObject( 0xa1a569278238dad2L );
}
```

存储持久数据

1. 导入以下类：
   - net.rim.device.api.system.PersistentStore
   - net.rim.device.api.system.PersistentObject
2. 在 PersistentObject 上调用 setContents()。此方法会将现有内容替换为新内容。
3. 要将新内容保存至持久存储区，请调用 commit()。

```
String[] userinfo = {username, password};
synchronized(store) {
store.setContents(userinfo);
store.commit();
}
```

4. 要使用批事务处理将对象提交至持久存储区，请调用 PersistentStore.getSynchObject()。此方法将检索锁定对象的持久存储区监视器。
   a. 在对象上同步。
   b. 根据需要调用 commit()。如果任何批提交都失败，则整个批事务处理都将失败。
5. 要单独从批事务处理提交监视器对象，请在同步监视器对象时调用 forceCommit()。

检索持久数据

1. 导入以下类：
   - net.rim.device.api.system.PersistentObject
   - net.rim.device.api.ui.component.Dialog
2. 在 PersistentObject 上调用 getContents()。
3. 要转换为所需格式，请在 PersistentObject.getContents() 返回的对象上执行显式转换。

```
synchronized(store) {
String[] currentinfo = (String[])store.getContents();
if(currentinfo == null) {
Dialog.alert(_resources.getString(APP_ERROR));
}
else {
```

```
currentusernamefield.setText(currentinfo[0]);
currentpasswordfield.setText(currentinfo[1]);
}
}
```

删除持久数据

如果删除定义 PersistentStore 的 .cod 文件，则将删除 .cod 文件创建的所有持久对象。

1. 导入以下类：
   - net.rim.device.api.system.PersistentStore
   - net.rim.device.api.system.PersistentObject
2. 要从 BlackBerry® 设备应用程序删除所有持久数据，请调用 PersistentStore.destroyPersistentObject()，其中将 PersistentObject 的唯一密钥作为参数提供。
3. 要删除单个数据，请将数据视为正常对象，并删除其引用。垃圾回收操作将删除数据。

MIDP 记录存储

创建 MIDP 记录存储区

1. 导入 javax.microedition.rms.RecordStore 类。
2. 调用 openRecordStore() 并指定 true，以指明如果记录存储区不存在，则此方法应该创建记录存储区。

```
RecordStore store = RecordStore.openRecordStore("Contacts", true);
```

将记录添加至记录存储区

1. 导入 javax.microedition.rms.RecordStore 类。
2. 调用 addRecord()。

```
int id = store.addRecord(_data.getBytes(), 0, _data.length());
```

从记录存储区检索记录

1. 导入以下类：
   - javax.microedition.rms.RecordStore
   - java.lang.String
2. 调用 getRecord(int, byte[], int)。传递以下参数：
   - 记录 ID

- 字节数组
- 偏移

```
byte[] data = new byte[store.getRecordSize(id)];
store.getRecord(id, data, 0);
String dataString = new String(data);
```

从记录存储区检索所有记录

1. 导入 javax.microedition.rms.RecordStore 类。
2. 导入以下接口：
   - javax.microedition.rms.RecordEnumeration
   - javax.microedition.rms.RecordFilter
   - javax.microedition.rms.RecordComparator
3. 调用 openRecordStore()。
4. 调用 enumerateRecords()。传递以下参数：
   - filter：指定 RecordFilter 对象以检索记录存储区记录的子集(如果为 null，方法将返回所有记录)
   - comparator：指定 RecordComparator 对象以确定方法返回记录的顺序(如果为 null，方法将按任何顺序返回记录)
   - keepUpdated：确定方法是否使用记录存储区更改更新枚举

```
RecordStore store = RecordStore.openRecordStore("Contacts", false);
RecordEnumeration e = store.enumerateRecords(null, null, false);
```

集合

从持久存储区检索集合

1. 导入以下类：
   - net.rim.device.api.system.PersistentStore
   - java.util.Vector
2. □□ net.rim.device.api.synchronization.SyncCollection 接口。
3. 要向 BlackBerry® 设备应用程序提供从 PersistentStore 访问最新 SyncCollection 数据的权限，请使用 SyncCollection 的 ID 调用 PersistentStore.getPersistentObject() 方法。

```
private PersistentObject _persist;
private Vector _contacts;
private static final long PERSISTENT_KEY = 0x266babf899b20b56L;
_persist = PersistentStore.getPersistentObject( PERSISTENT_KEY );
```

4. 将返回的数据存储在矢量对象中。

```
_contacts=(Vector)_persist.getContents();
```

5. 在无线数据备份会话开始之前，创建一个方法以向 BlackBerry 设备应用程序提供最新 SyncCollection 数据。

```
public void beginTransaction()
{
_persist = PersistentStore.getPersistentObject(PERSISTENT_KEY);
_contacts = (Vector)_persist.getContents();
}
```

6. 创建代码以管理从 PersistentStore 检索的 SyncCollection 为空的情况。

```
if( _contacts == null )
{
_contacts = new Vector();
_persist.setContents( _contacts );
_persist.commit();
}
```

创建集合监听器以在集合更改时通知系统

系统将调用 CollectionEventSource.addCollectionListener()，以为每个 SyncCollection 创建 CollectionListener。BlackBerry® 设备应用程序可用于无线备份。

1. 导入 net.rim.device.api.util.ListenerUtilities 类。
2. 导入以下接口：
   - java.util.Vector
   - net.rim.device.api.collection.CollectionEventSource
   - net.rim.device.api.collection.CollectionListener
   - net.rim.device.api.synchronization.SyncCollection
3. 创建私有矢量对象，以存储 BlackBerry 设备应用程序的 SyncCollection 监听器集合。

```
private Vector _listeners;
_listeners = new CloneableVector();
```

4. 实施 CollectionEventSource.addCollectionListener() 方法，其中确保方法会将 CollectionListener 添加至包含监听器的 Vector。在以下代码示例中，我们实施 CollectionEventSource.addCollectionListener() 以调用 ListenerUtilities.fastAddListener()，从而将监听器添加至包含监听器的Vector。

```
public void addCollectionListener(Object listener)
{
_listeners = ListenerUtilities.fastAddListener( _listeners, listener );
}
```

删除在集合更改时通知系统的集合监听器

不再需要某个 CollectionListener 时，系统将调用 CollectionEventSource.removeCollectionListener。

1. 导入以下类：
   - net.rim.device.api.util.ListenerUtilities
   - java.util.Vector
2. 导入以下接口：
   - net.rim.device.api.collection.CollectionEventSource
   - net.rim.device.api.collection.CollectionListener
3. 实施以下接口：
   - net.rim.device.api.collection.CollectionEventSource
   - net.rim.device.api.collection.CollectionListener
4. 实施 CollectionEventSource.removeCollectionListener() 方法，其中使用 ListenerUtilities.removeListener() 方法从包含 BlackBerry® 设备应用程序的 SyncCollection 监听器的 Vector 中删除 CollectionListener。在以下代码示例中，我们实施 CollectionEventSource.removeCollectionListener() 以调用 ListenerUtilities.removeListener()，从而从包含监听器的 Vector 中删除监听器。

```
public void removeCollectionListener(Object listener)
{_listeners = ListenerUtilities.removeListener( _listeners, listener );
}
```

在集合更改时通知系统

1. 导入 net.rim.device.api.collection.CollectionListener 接口。
2. 要在将元素添加至 SyncCollection 时通知系统，请调用 CollectionListener.elementAdded()。

```
for( int i=0; i<_listeners.size(); i++ )
{
CollectionListener cl = (CollectionListener)_listeners.elementAt( i );
cl.elementAdded( this, object );
}
return true;
}
```

3. 要在替换 SyncCollection 中的元素时通知系统，请调用 CollectionListener.elementUpdated()。
4. 调用 CollectionListener.elementRemoved()。

运行时存储

BlackBerry® 设备将运行时存储区用作 BlackBerry Java® Applications 可在其中共享运行时对象的中央位置。 默认情况下，只有 Research In Motion 数字签名的 BlackBerry Java Application 才能访问运行时存储区中的数据。 有关如何控制数据访问的信息，请联系 RIM。

运行时存储区非持久存储区。 重新启动 BlackBerry 设备时，将清除运行时存储区中的数据。

检索运行时存储区

1. 导入 net.rim.device.api.system.RuntimeStore 类。
2. 调用 RuntimeStore.getRuntimeStore()。

```
RuntimeStore store = RuntimeStore.getRuntimeStore();
```

在运行时存储区中添加对象

1. 导入以下类：
   - net.rim.device.api.system.RuntimeStore
   - java.lang.String
   - java.lang.IllegalArgumentException
2. 调用 RuntimeStore.put(long, String)，并将唯一长 ID 和运行时对象作为参数提供至存储区。
3. 创建 try - catch 块，以管理存在具有相同 ID 的运行时对象时 put() 将引发的 IllegalArgumentException。

```
RuntimeStore store = RuntimeStore.getRuntimeStore();
String msg = "Some shared text";
long ID = 0x60ac754bc0867248L;
try {
store.put( ID, msg );
} catch(IllegalArgumentException e) {
}
```

在运行时存储区中替换对象

1. 导入以下类：
   - net.rim.device.api.system.RuntimeStore
   - java.lang.String
   - net.rim.device.api.system.ControlledAccessException
2. 调用 replace()。

3. 创建 try - catch 块，以管理存在具有指定 ID 的运行时对象时 replace() 将引发的 ControlledAccessException。

```
RuntimeStore store = RuntimeStore.getRuntimeStore();
String newmsg = "Some new text";
try {
Object obj = store.replace( 0x60ac754bc0867248L, newmsg);
} catch(ControlledAccessException e) {
} not exist
```

检索注册的运行时对象

1. 导入以下类：
   - net.rim.device.api.system.RuntimeStore
   - net.rim.device.api.system.ControlledAccessException
2. 调用 RuntimeStore.get() 并将运行时对象 ID 作为参数提供。
3. 创建 try - catch 块，以管理 BlackBerry® Java® Application 无指定运行时对象的读取访问权限时 get() 将引发的 ControlledAccessException。

```
RuntimeStore store = RuntimeStore.getRuntimeStore();
try {
Object obj = store.get(0x60ac754bc0867248L);
} catch(ControlledAccessException e) {
}
```

检索未注册的运行时对象

1. 导入以下类：
   - net.rim.device.api.system.RuntimeStore
   - net.rim.device.api.system.ControlledAccessException
   - java.lang.RuntimeException
2. 调用 RuntimeStore.waitFor() 以等待运行时对象注册完成。
3. 创建代码以处理例外。

```
RuntimeStore store = RuntimeStore.getRuntimeStore();
try {
Object obj = store.waitFor(0x60ac754bc0867248L);
} catch(ControlledAccessException e) {
} catch(RuntimeException e) {
}
```

创建连接

3

网络连接和传输类型

BlackBerry® 设备可使用各种无线电通信技术，如 Wi-Fi® 技术、CDMA 或 GPRS，以打开无线连接。 无线连接将通过代理或网关传输至有线网络并连接至内部网络或 Internet。 BlackBerry 设备可使用各种类型的网关，且每个网关都提供唯一可配置功能集。 您可以指定要使用的无线连接和网关的类型。

运行 BlackBerry® Device Software 5.0 或更高版本的 BlackBerry 设备将包括旨在简化网络连接打开方式的网络 API，并将检查传输类型的可用性和无线网络覆盖区域。 通过指定 URL 和首选传输类型列表，可打开 HTTP、HTTPS 和套接字网络连接。 如果指定传输类型列表，网络 API 将按指定顺序检查每个传输类型的可用性和无线网络覆盖区域，并尝试打开连接。 此流程将继续，直至打开网络连接或到达列表末尾。 如果未指定传输类型，将尝试所有可用传输类型。

网络 API 的大部分功能都在 `ConnectionFactory` 和 `TransportInfo` 类中实施。 您可以使用 `TransportInfo` 类中提供的方法检查可用传输类型及关联无线网络覆盖区域。 您可以使用 `ConnectionFactory` 类请求网络连接。 所有网络 API 类都在 `net.rim.device.api.io.transport` 和 `net.rim.device.api.io.transport.options` 数据包中提供。

将 BlackBerry Enterprise Server 用作内部网络网关

企业客户可在其公司防火墙后托管 BlackBerry® Enterprise Server，以便可从 BlackBerry 设备访问公司内部网络。 BlackBerry Enterprise Server 的 BlackBerry® Mobile Data System 组件包括 BlackBerry® MDS Services，其提供的 HTTP 和 TCP/IP 代理服务允许第三方 Java® 应用程序将其用作管理至内部网络的 HTTP 和 TCP/IP 连接的安全网关。 将 BlackBerry Enterprise Server 用作内部网络网关时，应用程序和 BlackBerry Enterprise Server 之间的流量都使用 AES 或 Triple DES 加密自动加密。 因为 BlackBerry Enterprise Server 位于组织防火墙后且提供固有数据加密，所以应用程序可与位于公司内部网络上的应用程序服务器和 Web 服务器通信。

如果应用程序连接至 Internet，而不是公司内部网络，可将属于客户的 BlackBerry Enterprise Server 用作网关。 在这种情况下，网络请求通过组织防火墙后并到达 BlackBerry Enterprise Server，这让网络请求可通过公司防火墙到达 Internet。 但是，企业客户可设置 IT 策略，以强制 BlackBerry Enterprise Server 是用于所有无线网络流量的网关，包括至 Internet 的流量。

如果应用程序连接至 Internet，且您面向非企业客户，还可使用 BlackBerry® Internet Service 或无线服务提供商的 Internet 网关管理连接。

使用无线服务提供商的 Internet 网关

用于 BlackBerry® 设备的 Java® 应用程序可使用无线服务提供商提供的 Internet 网关连接至 Internet。 大部分无线服务提供商提供的自己 Internet 网关都可提供至 Internet 的直接 TCP/IP 连接。 一些运营商提供的 WAP 网关还允许通过 WAP 协议进行 HTTP 连接。 用于 BlackBerry 设备的 Java 应用程序可使用其中一个网关建立 Internet 连接。 如果编写的应用程序面向位于特定无线网络的 BlackBerry 设备用户，此方法通常可获得良好的效果。 但是，如果编写的应用程序面向位于各种无线网络的 BlackBerry 设备用户，则根据各种 Internet 网关测试应用程序和实现一致且可靠的体验可能具有挑战性。 在这种情况下，您可能会发现使用 BlackBerry® Internet Service 且在 BlackBerry Internet Service 不可用时将无线服务提供商的 Internet 网关用作默认连接类型很有用。

检索无线网络名称

1. 导入以下类：
   - net.rim.device.api.system.RadioInfo
   - java.lang.String
   - net.rim.device.api.ui.Field
2. 调用 RadioInfo.getCurrentNetworkName()。BlackBerry® 设备必须连接至无线网络,此方法才能正常运行。

```
String networkName = RadioInfo.getCurrentNetworkName();
System.out.println (“Network Name: “ + networkName );
```

连接

使用 HTTP 身份验证

1. 导入以下类：
   - net.rim.device.api.system.CoverageInfo
   - javax.microedition.io.Connector
   - net.rim.device.api.ui.UiApplication
   - net.rim.device.api.ui.component.Dialog
   - java.lang.String
2. 导入以下接口：
   - javax.microedition.io.HttpConnection
   - net.rim.device.api.system.CoverageStatusListener
   - javax.microedition.io.StreamConnection

3. 使用 net.rim.device.api.system 数据包的 CoverageInfo 类和 CoverageStatusListener 接口，以确定 BlackBerry 设备位于无线网络覆盖区域中。
4. 调用 Connector.open()，其中使用受保护资源的 HTTP 位置。
5. 将返回的对象转换并存储为 StreamConnection。

```
StreamConnection s = (StreamConnection)Connector.open("http://mysite.com/
myProtectedFile.txt");
```

6. 将 StreamConnection 对象转换并存储为 HTTPConnection 对象。

```
HttpConnection httpConn = (HttpConnection)s;
```

7. 调用 HttpConnection.getResponseCode()。

```
int status = httpConn.getResponseCode();
```

8. 创建代码以管理未授权的 HTTP□□ 尝试。

```
int status = httpConn.getResponseCode();
switch (status)
case (HttpConnection.HTTP_UNAUTHORIZED);
```

9. 创建 run()method 并在其中实施 dialog 对象，以提示 BlackBerry 设备用户输入登录信息。

```
UiApplication.getUiApplication().invokeAndWait(new Runnable())
{
public void run()
{
dialogResponse = Dialog.ask;
(Dialog.D_YES_NO,"Unauthorized Access:\n Do you wish to log in?");
}
}
```

10. 要处理登录信息，请创建代码以管理 Yes 对话框响应。
    a. 检索登录信息并关闭当前连接。

    ```
    if (dialogResponse == Dialog.YES)
    {String login = "username:password";
    //Close the connection.
    s.close();
    ```

    b. 将登录信息编码。

    ```
    byte[] encoded = Base64OutputStream.encode(login.getBytes(),
    0, login.length(), false, false);
    ```

11. 使用已编码的登录信息调用 HTTPConnection.setRequestProperty()，以访问受保护资源。

```
s = (StreamConnection)Connector.open("http://mysite.com/myProtectedFile.txt ");
httpConn = (HttpConnection)s;
httpConn.setRequestProperty("Authorization", "Basic " + new String(encoded));
```

使用 HTTPS 连接

如果 BlackBerry 设备与 BlackBerry® Enterprise Server 关联且使用需要身份验证的 HTTPS 代理服务器，将无法使用端对端 TLS.

1. 导入以下类：
   - `net.rim.device.api.system.CoverageInfo`
   - `javax.microedition.io.Connector`
2. 导入以下接口：
   - `net.rim.device.api.system.CoverageStatusListener`
   - `javax.microedition.io.HttpsConnection`
3. 使用 `net.rim.device.api.system` 数据包的 `CoverageInfo` 类和 `CoverageStatusListener` 接口,以确保 BlackBerry 设备位于无线网络覆盖区域中。
4. 调用 `Connector.open()`（其中将 HTTPS 指定为协议),然后将返回的对象转换为 `HttpsConnection` 对象,以打开 HTTP 连接。

```
HttpsConnection stream = (HttpsConnection)Connector.open("https://host:443/");
```

5. 要指定连接模式,请将以下参数之一添加至传递至 `Connector.open()` 的连接字符串
   - 指定必须使用从 BlackBerry 设备至目标服务器的端对端 HTTPS 连接：`EndToEndRequired`
   - 指定应该使用从 BlackBerry 设备至目标服务器的端对端 HTTPS 连接。如果 BlackBerry 设备不支持端对端 TLS,且 BlackBerry 设备用户允许代理 TLS 连接,则使用代理连接：`EndToEndDesired`。

```
HttpsConnectionstream=(HttpsConnection)Connector.open("https://host:
443/;EndToEndDesired");
```

使用套接字连接

尽管可通过套接字连接实施 HTTP，但出于以下原因应该使用 HTTP 连接：
- 套接字连接不支持 BlackBerry® Mobile Data System 功能,如推入。
- 使用套接字连接的 BlackBerry® 设备应用程序通常需要比使用 HTTP 连接的 BlackBerry 设备应用程序多很多的带宽。

1. 导入以下类：
   - `net.rim.device.api.system.CoverageInfo`
   - `javax.microedition.io.Connector`
   - `java.lang.String`
   - `java.io.OutputStreamWriter`
   - `java.io.InputStreamReader`
2. 导入以下接口：
   - `net.rim.device.api.system.CoverageStatusListener`

- `javax.microedition.io.StreamConnection`

3. 使用 `net.rim.device.api.system` 数据包的 `CoverageInfo` 类和 `CoverageStatusListener` 接口，以确保 BlackBerry 设备位于无线网络覆盖区域中。
4. 调用 `Connector.open()`，其中将 **socket** 指定为协议并将 *deviceside*=false 参数附加至 URL 结尾。
   - 要使用 BlackBerry MDS Services 打开套接字连接，请将 `deviceside=false` 附加至 URL 结尾。BlackBerry 设备应用程序必须显式输入本地计算机 IP，因为 `localhost` 不受支持。

     ```
     private static String URL = "socket://local_machine_IP:4444;deviceside=false";
     StreamConnection conn = null;
         conn = (StreamConnection)Connector.open(URL);
     ```
   - 要通过直接 TCP 打开套接字连接，请将 `deviceside=true` 参数附加至 URL 结尾。

     ```
     private static String URL = "socket://local_machine_IP:4444;deviceside=true";
     StreamConnection conn = null;
     conn = (StreamConnection)Connector.open(URL);
     ```
   - 要通过直接 TCP 打开套接字连接，请指定 APN 信息，将 *deviceside*=true 参数附加至 URL 结尾，然后指定将用于建立连接的 APN。指定要连接至 APN 的用户名和密码（如果 APN 需要）。

   ```
   private static String URL = "socket:
   //local_machine_IP:4444;deviceside=true;apn=internet.com;tunnelauthusername
   =user165;tunnelauthpassword=user165password";
   StreamConnection conn = null;
   conn = (StreamConnection)Connector.open(URL);
   ```
5. 使用 `openInputStream()` 和 `openOutputStream()` 收发数据。

   ```
   OutputStreamWriter _out = new OutputStreamWriter(conn.openOutputStream());
   String data = "This is a test";
   int length = data.length();
   _out.write(data, 0, length);
   InputStreamReader _in = new InputStreamReader(conn.openInputStream());
   char[] input = new char[length];
   for ( int i = 0; i < length; ++i ) {
   input[i] = (char)_in.read();
   };
   ```
6. 在输入流和输出流及套接字连接上调用 `close()`。每个 `close()` 方法都将引发 `IOException`。确保 BlackBerry 设备应用程序实施例外处理。

   ```
   _in.close();
   _out.close();
   conn.close();
   ```

使用数据报连接

数据报是应用程序通过网络发送的数据的独立数据包。 `Datagram` 对象是为数据报的有效负载的字节数组的包装程序。 使用数据报连接可收发数据报。

要使用数据报连接，必须将自己的基础架构连接至无线网络，包括 GPRS 网络的 APN。 要使用 UDP 连接，需要与服务提供商紧密合作。 确定服务提供商支持 UDP 连接。

1. 导入以下类和接口：
   - net.rim.device.api.system.CoverageInfo
   - javax.microedition.io.Connector
   - java.lang.String
2. 导入以下接口：
   - net.rim.device.api.system.CoverageStatusListener
   - javax.microedition.io.DatagramConnection
   - javax.microedition.io.Datagram
3. 使用 net.rim.device.api.system 数据包的 CoverageInfo 类和 CoverageStatusListener 接口,以确保 BlackBerry 设备位于无线网络覆盖区域中。
4. 调用 Connector.open(),将 udp 指定为协议,然后将返回的对象转换为 DatagramConnection 对象，以打开数据报连接。

   ```
   (DatagramConnection)Connector.open("udp://host:dest_port[;src_port]/apn");
   ```

   其中：

   - **host** 是以点 ASCII 十进制格式表示的主机地址。
   - **dest-port** 是位于主机地址的目标端口(接收消息时为可选)。
   - **src-port** 是本地源端口(可选)。
   - **apn** 是以字符串格式表示的网络 APN。
5. 要从指定主机的所有端口接收数据报,请忽略连接字符串中的目标端口。
6. 要在非 GPRS 网络上打开数据报连接,请指定源端口号,包括尾随斜线。例如,CDMA 网络连接的地址将为 udp://121.0.0.0:2332;6343/。您可在同一端口上收发数据报。
7. 要创建数据报,请调用 DatagramConnection.newDatagram()。

   ```
   Datagram outDatagram = conn.newDatagram(buf, buf.length);
   ```
8. 要将数据添加至图中,请调用 Datagram.setData()。

   ```
   byte[] buf = new byte[256];
   outDatagram.setData(buf, buf.length);
   ```
9. 要通过数据报连接发送数据,请在数据报连接上调用 send()。

   ```
   conn.send(outDatagram);
   ```

   如果 BlackBerry®Java® Application 尝试通过数据报连接发送数据,且收件人未在监听指定源端口,将引发 IOException。确保 BlackBerry Java Application 实施例外处理。
10. 要通过数据报连接接收数据,请在数据报连接上调用 receive()。receive() 方法将阻止其它操作,直至收到数据包。如果未收到答复,则使用计时器重新传输请求或关闭连接。

```
byte[] buf = new byte[256];
Datagram inDatagram = conn.newDatagram(buf, buf.length);
conn.receive(inDatagram);
```

11. 要从数据报提取数据，请调用 getData()。如果知道要接收的数据类型，请将数据转换为相应格式。

```
String received = new String(inDatagram.getData());
```

12. 关闭数据报连接，在输入流和输出流上调用 close() ，然后在数据报连接对象上调用。

```
conn.close();
```

使用 USB 或串行端口连接

使用串行或 USB 连接时，BlackBerry® 设备应用程序在使用串行或 USB 端口连接至计算机时，可与桌面应用程序通信。 这种连接类型还让 BlackBerry 设备应用程序可与插入串行或 USB 端口的外围设备通信。

1. 导入以下类：
   - javax.microedition.io.Connector
   - java.io.DataOutputStream
   - java.lang.String
   - java.io.DataInputStream
2. 导入 javax.microedition.io.StreamConnection 接口。
3. 调用 Connector.open()，然后将 **comm** 指定为协议，并将 COM1 或 USB 指定为端口，以打开 USB 或串行端口连接。

```
private StreamConnection _conn = (StreamConnection)Connector.open(
"comm:COM1;baudrate=9600;bitsperchar=8;parity=none;stopbits=1");
```

4. 要通过 USB 或串行端口连接发送数据，请调用 openDataOutputStream() 或 openOutputStream()。

```
DataOutputStream _dout = _conn.openDataOutputStream();
```

5. 在输出流上使用写入方法写入数据。

```
private String data = "This is a test";
_dout.writeChars(data);
```

6. 要通过 USB 或串行端口连接接收数据，请使用非主事件线程从输入流读取数据。调用 openInputStream() 或 openDataInputStream()。

```
DataInputStream _din = _conn.openInputStream();
Use the read methods on the input stream to read data.
```

7. 在输入流上使用读入方法读取数据。

```
String contents = _din.readUTF();
```

8. 要关闭 USB 或串行端口连接，请在输入流和输出流上调用 `close()`，然后在端口连接对象上调用。`close()` 方法可引发 `IOException`。确保 BlackBerry 设备应用程序实施例外处理。

```
_din.close();
_dout.close();
conn.close();
```

使用 Bluetooth 是 Bluetooth SIG 的商标。串行端口连接

您可以使用 Bluetooth 是 Bluetooth SIG 的商标。 API (`net.rim.device.api.bluetooth`)，以让 BlackBerry® 设备应用程序可访问 Bluetooth 是 Bluetooth SIG 的商标。串行端口配置文件，并可启动至计算机或其它 Bluetooth 是 Bluetooth SIG 的商标。设备的服务器或客户端 Bluetooth 是 Bluetooth SIG 的商标。串行端口连接。

1. 导入以下类：
   - `javax.microedition.io.Connector`
   - `net.rim.device.api.bluetooth.BluetoothSerialPort`
   - `java.io.DataOutputStream`
   - `java.io.DataInputStream`
   - `java.lang.String`
   - `java.io.IOException`
2. 导入 `javax.microedition.io.StreamConnection` 接口。
3. 调用 `Connector.open()` 以打开 Bluetooth 是 Bluetooth SIG 的商标。连接，其中将 `BluetoothSerialPort.getSerialPortInfo()` 返回的串行端口信息作为参数提供。

```
BluetoothSerialPortInfo[] info = BluetoothSerialPort.getSerialPortInfo();
StreamConnection _bluetoothConnection = (StreamConnection)Connector.open( info
[0].toString(), Connector.READ_WRITE );
```

4. 要通过 Bluetooth 是 Bluetooth SIG 的商标。连接发送数据，请调用 `openDataOutputStream()` or `openOutputStream()`。

```
DataOutputStream _dout = _bluetoothConnection.openDataOutputStream();
```

5. 在输出流上使用写入方法写入数据。

```
private static final int JUST_OPEN = 4;
_dout.writeInt(JUST_OPEN);
```

6. 要通过 Bluetooth 是 Bluetooth SIG 的商标。连接接收数据，请在非主事件线程中调用 `openInputStream()` 或 `openDataInputStream()`。在输入流上使用读入方法读取数据。

```
DataInputStream _din = _bluetoothConnection.openDataInputStream();
String contents = _din.readUTF();
```

7. 在输入流和输出流上调用 `close()`，并在 Bluetooth 是 Bluetooth SIG 的商标。串行端口连接对象上调用，以关闭 Bluetooth 是 Bluetooth SIG 的商标。连接。`close()` 方法可引发 `IOException`。确保 BlackBerry 设备应用程序实施例外处理。

```
if (_bluetoothConnection != null) {
try {
      _bluetoothConnection.close();
    } catch(IOException ioe) {
      }
}
if (_din != null) {
try {
      _din.close();
      } catch(IOException ioe) {
        }
}
if (_dout != null) {
try {
      _dout.close();
    } catch(IOException ioe) {
      }
}
_bluetoothConnection = null;
_din = null;
_dout = null;
```

Wi-Fi 连接

无线接入系列

使用 BlackBerry® 设备收发器涉及使用引用无线接入系列的 API。

无线接入系列	说明
3GPP	包括 GPRS、EDGE、UMTS®、GERAN、UTRAN 和 GAN
CDMA	包括 CDMA1x 和 EVDO
WLAN	包括 802.11™、802.11a™、802.11b™、802.11g™

有关无线接入系列的详细信息，请参阅 BlackBerry® Java® Development Environment 的 API 参考

检索 BlackBerry 设备支持的无线接入系列

1. 导入 `net.rim.device.api.system.RadioInfo` 类。

2. 调用 RadioInfo.getSupportedWAFs()。

确定 BlackBerry 设备是否支持多个无线接入系列

1. 导入 net.rim.device.api.system.RadioInfo 类。
2. 调用 RadioInfo.areWAFsSupported(intwafs)。

确定打开的无线接入系列收发器

1. 导入 net.rim.device.api.system.RadioInfo 类。
2. 调用 RadioInfo.getActiveWAFs()。

打开无线接入系列的收发器

1. 导入 net.rim.device.api.system.Radio 类。
2. 调用 Radio.activateWAFs(intWAFs)。WAFs 参数是位掩码。

关闭无线接入系列的收发器

1. 导入 net.rim.device.api.system.Radio 类。
2. 调用 Radio.deactivateWAFs(intWAFs)。WAFs 参数是位掩码。

检查是否打开了 Wi-Fi 收发器

1. 导入 net.rim.device.api.system.RadioInfo 类。
2. 创建测试 RadioInfo.WAF_WLAN 的值和 RadioInfo.getActiveWAFs() 返回的值的 IF 语句。

```
if ( ( RadioInfo.getActiveWAFs() & RadioInfo.WAF_WLAN ) != 0 ) { … }
```

检查 Wi-Fi 收发器是否连接至无线接入点

1. 导入 net.rim.device.api.system.WLANInfo 类。
2. 创建测试 WLANInfo.WLAN_STATE_CONNECTED 的值和 WLANInfo.getWLANState() 返回的值的 IF 语句。WLANInfo.getWLANState() 方法检查 BlackBerry® 设备是否具有 IP 地址以及是否可通过 Wi-Fi® 网络传输数据。如果 WLAN 无线接入系列的收发器关闭,此方法将返回 WLANInfo.WLAN_STATE_DISCONNECTED。

```
    if (WLANInfo.getWLANState() == WLANInfo.WLAN_STATE_CONNECTED) {…}
```

检索无线接入点或活动 Wi-Fi 配置文件的状态

您可以让 BlackBerry® 设备应用程序检索状态信息，如连接的数据传输速率、使用的无线 LAN 标准（802.11a™、802.11b™ 或 802.11g™)、关联接入点的 SSID 或使用的 Wi-Fi 配置文件的名称。 WLAN 无线接入系列的收发器必须连接至无线接入点。

1. 导入 `net.rim.device.api.system.WLANInfo` 类。
2. 调用 `WLANInfo.getAPInfo()`,其中存储此方法返回的 `WLANInfo.WLANAPInfo` 的引用。`WLANInfo.WLANAPInfo` 对象包含当前无线网络的快照。

```
    WLANInfo.WLANAPInfo info = WLANInfo.getAPInfo();
```

   如果 BlackBerry 设备未连接至接入点,则 `WLANInfo.getAPInfo()` 方法返回 null。

检索无线网络名称

1. 导入以下类:
   - `net.rim.device.api.system.RadioInfo`
   - `java.lang.String`
   - `net.rim.device.api.ui.Field`
2. 调用 `RadioInfo.getCurrentNetworkName()`。BlackBerry® 设备必须连接至无线网络,此方法才能正常运行。

```
String networkName = RadioInfo.getCurrentNetworkName();
System.out.println (“Network Name: “ + networkName );
```

打开 Wi-Fi 套接字连接

`interface=wifi` 参数仅适用于 TCP/UDP 连接。 要建立 Wi-Fi® 连接并将 Wi-Fi API 用于 BlackBerry® 设备应用程序，无线服务提供商必须支持 Wi-Fi 访问。

1. 导入以下类:
   - `java.lang.String`
   - `javax.microedition.io.Connector`
2. 导入 `javax.microedition.io.StreamConnection` 接口。
3. 调用 `Connector.open()`,将 **socket** 指定为协议,然后将 `deviceside=true` 参数和 `interface=wifi` 参数附加至 URL 字符串值的末尾。

```
private static String URL = "socket://local_machine_IP:
4444;deviceside=true;interface=wifi";
StreamConnection conn = null;
conn = (StreamConnection)Connector.open(URL);
```

打开 Wi-Fi HTTP 连接

interface=wifi 参数仅适用于 TCP/UDP 连接。 要建立 Wi-Fi® 连接并将 Wi-Fi API 用于 BlackBerry® 设备应用程序，无线服务提供商必须支持 Wi-Fi 访问。

1. 导入以下类：
   - java.lang.String
   - javax.microedition.io.Connector
2. 导入 javax.microedition.io.HttpConnection 接口。
3. 调用 Connector.open(),将 **http** 指定为协议,然后将 interface=wifi 参数附加至 URL 字符串值的末尾。
4. 将返回的对象转换为 HttpConnection 或 StreamConnection 对象。

```
HttpConnection conn = null;
String URL = "http://www.myServer.com/myContent;deviceside=true;interface=wifi";
conn = (HttpConnection)Connector.open(URL);
```

打开 Wi-Fi HTTPS 连接

interface=wifi 参数仅适用于 TCP/UDP 连接。 要建立 Wi-Fi® 连接并将 Wi-Fi API 用于 BlackBerry® 设备应用程序，无线服务提供商必须支持 Wi-Fi 访问。

1. 导入以下类：
   - java.lang.String
   - javax.microedition.io.Connector
2. 导入 javax.microedition.io.HttpsConnection 接口。
3. 调用 Connector.open(),将 **http**s 指定为协议,然后将 interface=wifi 参数附加至 URL 字符串值的末尾。
4. 将返回的对象转换为 HttpsConnection 对象。

```
HttpsConnection conn = null;
String URL = "https://host:443/;deviceside=true;interface=wifi";
conn = (HttpsConnection)Connector.open(URL);
```

增强网络 API

通过使用 net.rim.device.api.io.transport 数据包和 net.rim.device.api.io.transport.options 数据包提供的增强网络 API，可指定应用程序打开无线网络连接的方式。 网络 API 可简化应用程序建立无线网络连接的方式，并让您可以找到 BlackBerry® 设备上可用的无线传输类型。 例如，可使用网络 API 找到应用程序可用于打开 HTTP 连接的 Wi-Fi® 网络的可用性和信号强度。

您可使用网络 API 指定无线传输类型的首选列表。 例如，如果 Wi-Fi 网络不可用或信号强度不足，则可让应用程序搜索其它传输类型，如蜂窝网络。 如果未指定传输类型，默认情况下，BlackBerry 设备将搜索所有可用传输类型。

有关网络 API 的详细信息，请参阅 *BlackBerry Java Application Development Guide》* 及 BlackBerry® Java® Development Environment 的 API 参考。

代码示例：显示可用运输类型

```
/*
* DisplayAvailableTransportTypesScreen.java
*
* Research In Motion Limited proprietary and confidential
* Copyright Research In Motion Limited, 2009
*/
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.io.transport.*;
import java.lang.StringBuffer;
public class DisplayAvailableTransportTypes extends UiApplication
{
   public static RichTextField _message = new RichTextField("");
public static void main(String[] args)
{
    DisplayAvailableTransportTypes theApp = new DisplayAvailableTransportTypes();
    theApp.enterEventDispatcher();
}
public DisplayAvailableTransportTypes()
{
    pushScreen(new DisplayAvailableTransportTypesScreen());
}
}
class DisplayAvailableTransportTypesScreen extends MainScreen
{
    private RichTextField strMessage = new RichTextField("");
    public DisplayAvailableTransportTypesScreen()
```

```
    {
        LabelField title = new LabelField("Transport Types Sample",
        LabelField.ELLIPSIS |
        LabelField.USE_ALL_WIDTH);
        setTitle(title);
        add(new RichTextField("Detected the following transport types:\n"));
            int TransportTypes[] = TransportInfo.getAvailableTransportTypes();
            StringBuffer sbTmp = new StringBuffer("");
            for(int i=0; i<TransportTypes.length; i++)
            {
                int tt = TransportTypes[i];
                switch(tt)
                {
                    case TransportInfo.TRANSPORT_BIS_B:

sbTmp.append(" BIS-B\n"); break;
                    case TransportInfo.TRANSPORT_MDS:

sbTmp.append("* MDS\n");  break;
                    case TransportInfo.TRANSPORT_TCP_CELLULAR:

sbTmp.append("* TCP Cellular\n"); break;
                    case TransportInfo.TRANSPORT_TCP_WIFI:

sbTmp.append("* Wifi\n"); break;
                    case TransportInfo.TRANSPORT_WAP:

sbTmp.append("* WAP 1.0 or 1.1\n"); break;
                    case TransportInfo.TRANSPORT_WAP2:

sbTmp.append("* WAP 2.0\n"); break;
                }
            }
            strMessage.setText(sbTmp.toString());
            add(strMessage);
    }
}
```

代码示例：通过首个可用传输通过 HTTP 创建连接

```
/*
* HTTPFirstAvailable.java
*
* Research In Motion Limited proprietary and confidential
* Copyright Research In Motion Limited, 2010
*/
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import net.rim.device.api.io.transport.*;
```

```
import javax.microedition.io.*;
import java.io.*;
public class HTTPFirstAvailable extends UiApplication
{
    public static void main(String[] args)
    {
        HTTPFirstAvailable theApp = new HTTPFirstAvailable();
        theApp.enterEventDispatcher();
    }
    public HTTPFirstAvailable()
    {
        pushScreen(new HTTPFirstAvailableScreen());
    }
}
class ConnectionThread extends Thread
{
    public void run()
    {
          ConnectionFactory connFact = new ConnectionFactory();
          ConnectionDescriptor connDesc;
          connDesc = connFact.getConnection("http://www.example.com");
          if (connDesc != null)
          {
              HttpConnection httpConn;
              httpConn = (HttpConnection)connDesc.getConnection();
              try
              {
                  final int iResponseCode = httpConn.getResponseCode();
                  UiApplication.getUiApplication().invokeLater(new Runnable()
                  {
                      public void run()
                      {
                          Dialog.alert("Response code: " +
                                        Integer.toString(iResponseCode));
                      }
                   });
               }
               catch (IOException e)
               {
                 System.err.println("Caught IOException: "
                      + e.getMessage());
               }
          }
    }
}
class HTTPFirstAvailableScreen extends MainScreen
{
    public HTTPFirstAvailableScreen()
    {
        setTitle("HTTP First Sample");
        add(new RichTextField("Trying to make HTTP connection... \n"));
        ConnectionThread ct = new ConnectionThread();
```

```
        ct.start();
    }
}
```

控制 API 和应用程序数据访问 4

检查是否需要代码签名

Research In Motion 会因安全和导出控制原因跟踪敏感 API 在 BlackBerry® Java® Development Environment 中的使用。

在 BlackBerry Java Development Environment 的 API 参考中找到该项目。如果该项目具有锁图标或标注为"已签名",则 BlackBerry 设备应用程序需要 RIM 提供的已签名密钥或签名,才能将 BlackBerry 设备应用程序 .cod 文件加载至 BlackBerry 设备上。

Java API 具有受控访问

RIM 控制运行时 API、BlackBerry® 应用程序 API 和 BlackBerry 加密法 API。

无代码签名也可在 BlackBerry® Smartphone Simulator 中测试使用受控 API 的 BlackBerry 设备应用程序；但是，必须先从 RIM 获取代码签名，才能将这些 BlackBerry 设备应用程序加载至 BlackBerry 设备上。

如果使用任何以下 BlackBerry API 数据包，BlackBerry 应用程序需要代码签名，才能将其加载至 BlackBerry 设备上：

- net.rim.blackberry.api.browser
- net.rim.blackberry.api.invoke
- net.rim.blackberry.api.mail
- net.rim.blackberry.api.mail.event
- net.rim.blackberry.api.menuitem
- net.rim.blackberry.api.options
- net.rim.blackberry.api.pdap
- net.rim.blackberry.api.phone
- net.rim.blackberry.api.phone.phonelogs
- net.rim.device.api.browser.field
- net.rim.device.api.browser.plugin
- net.rim.device.api.crypto
- net.rim.device.api.io.http
- net.rim.device.api.notification
- net.rim.device.api.servicebook
- net.rim.device.api.synchronization
- net.rim.device.api.system

注册使用受控 API

1. 在以下网址填写注册表单 https://www.blackberry.com/SignedKeys/。
2. 保存 Research In Motion 通过电子邮件发送给您的 .csi 文件。.csi 文件包含签名和注册信息的列表。如果 BlackBerry® Signing Authority Tool 管理员未向您提供 .csi 文件或客户端 PIN,且您是 ISV 合作伙伴,请与 ISV 技术合作伙伴关系经理联系。如果您不是 ISV 合作伙伴,请将电子邮件发送至 jde@rim.com。
3. 双击 .csi 文件。
4. 如果显示的对话框声称无法找到私钥,请先完成步骤 5 至 8,再继续。否则,请继续执行步骤 9。
5. 单击**是**以创建新密钥对文件。
6. 在**私钥密码**字段中,键入至少包含 8 个字符的密码,然后再次键入以确认。私钥密码保护私钥。如果丢失此密码,必须再次在 RIM 中注册。如果此密码被偷,请立即与 RIM 联系。
7. 单击**确定**。
8. 移动鼠标为新私钥生成数据。
9. 在**注册 PIN** 字段中,键入 RIM 提供的 **PIN**。
10. 在**私钥密码**字段中,键入**私钥**密码。
11. 单击**注册**。
12. 单击**退出**。

代码签名限制

BlackBerry® Signing Authority Tool 管理员可对 .csi 文件进行限制，以限制代码签名访问。 要请求更改这些限制，请与管理员联系。

.csi 文件限制	说明
请求次数	此限制指定可使用特定 .csi 文件进行的最大请求次数。 达到最大请求次数时，.csi 文件将无效。 要进行新代码签名请求，必须申请新 .csi 文件。 尽管管理员可允许无限的请求次数，但出于安全原因，通常将请求次数指定为有限次数。
到期日期	此限制指定 .csi 文件的到期日期。 在到期日期之后，不能再使用此 .csi 文件申请代码签名。 要进行新签名请求，必须申请新 .csi 文件。

请求代码签名

BlackBerry® Signature Tool 包括在 BlackBerry® Java® Development Environment 中。 BlackBerry JDE 可从以下位置下载： www.blackberry.com/developers。 安装 BlackBerry® Signing Authority Tool 时，将安装 Web Signer 应用程序。

有关 Web Signer 应用程序的详细信息，请参阅《BlackBerry Signing Authority Tool version 1.0 - Password Based Administrator Guide》。

开始之前：您必须从 Research In Motion 获取 .csi 文件。

1. 在 Windows® Internet Explorer® 中，找到要为其请求签名的 BlackBerry 设备应用程序的 .cod 文件。
2. 确保在 .cod 文件所在的文件夹中存在与 .cod 文件的名称相同的 .csl 文件。BlackBerry® Integrated Development Environment 编译器将自动生成 .csl 文件。
3. 双击 .cod 文件以将其添加至签名列表。签名列表包含有关要拥有其访问权限且要为其请求签名的 .cod 文件的信息。
4. 对于每个要添加至签名列表的 .cod 文件，重复步骤 1 至 3。
5. 在 BlackBerry Signature Tool 菜单中，单击**请求**。
6. 在对话框中，键入私钥密码。
7. 单击**确定**。

BlackBerry Signature Tool 使用私钥密码将签名附加至请求，并将 .cod 文件的签名列表添加至 Web Signer 应用程序，以便验证。

使用代理服务器注册签名密钥

每个 .csi 文件都只能注册一次。

1. 在命令提示符处，导航至 BlackBerry® Signature Tool bin 目录。例如：
   C:\Program Files\Research In Motion\BlackBerry JDE 4.6.0\bin
2. 使用以下参数键入 Java -jar -Dhttp.proxyHost=myproxy.com -Dhttp.proxyPort=80 SignatureTool.jar SigKey.csi：
   - SigKey：每个签名密钥（.csi）文件的名称。将以下命名约定用于密钥：client-RRT-*.csi client-RBB-*.csi client-RCR-*.csi
   - Dhttp.proxyHost：代理服务器的名称或 IP 地址。
   - Dhttp.proxyPort：如果未将 80 指定为默认端口号，则为代理服务器端口号。
3. 对于每个要注册的 .csi 文件，重复步骤 2。

使用代理服务器对应用程序进行签名

注册密钥和 .csk 文件将存储在一起。 如果丢失注册密钥或 .csk 文件，则无法请求代码签名。 如果您不是 ISV 合作伙伴，请与 ISV 技术合作伙伴关系经理联系。 如果您是非 ISV 合作伙伴，请将电子邮件发送至 jde@rim.com。

1. 在命令提示符处,导航至 BlackBerry® Signature Tool bin 目录。例如：
   C:\Program Files\Research In Motion\BlackBerry JDE 4.6.0\bin
2. 键入 Java -jar -Dhttp.proxyHost=myproxy.com -Dhttp.proxyPort=80 SignatureTool.jar
3. 在“文件选择”窗口中,选择要签名的 .cod 文件。
4. 单击**打开**。

查看应用程序的签名状态

对于未签名的文件，**状态**列包含**失败**。 Web Signer 可能拒绝了 .cod 文件，因为键入的私钥密码错误。

1. 启动 BlackBerry® Signature Tool。
2. 选择 .cod 文件。
3. 查看**状态**列。
   对于 Web Signer 已签名的文件,**状态**列包含**已签名**。

使用密钥保护 API 和数据

要创建用于内部签名授权系统的内部密钥对，或要创建用于内部签名授权系统的外部密钥对，必须将保护应用于敏感 API，保护运行时存储区中的数据，以及保护持久对象中的数据。

RSAE.key 是外部密钥，而 ACMI.key 是内部密钥。

使用代码签名密钥保护 API

1. 收到内部密钥、外部密钥或同时收到两者后,请在 BlackBerry® Integrated Development Environment 中打开包含要控制其访问的 API 的项目。
2. 在“工作区”窗口中,右键单击项目文件。
3. 单击**将文件添加至项目**。
4. 在**搜索范围**字段中,浏览至 C:\Program Files\Research In Motion\BlackBerry Password Based Code Signing Authority\data 或 .key 文件的保存位置。
5. 请执行以下任务之一：

选项	说明
使用内部密钥	a. 选择内部 .key 文件，如 **ACMI.key** 文件。 b. 单击**打开**。 c. 在“工作区”窗口中，右键单击 .key 文件。 d. 选择**用作公共类的默认值**选项和**用作非公共类的默认值**设置。 e. 单击**确定**。 f. 在“工作区”窗口中，右键单击项目文件。 g. 单击**将文件添加至项目**。 h. 在**搜索范围**字段中，浏览至 C:\Program Files \Research In Motion\BlackBerry Password Based Code Signing Authority\data。
使用外部密钥	a. 选择外部 .key 文件，如 **RSAE.key** 文件。 b. 单击**打开**。 c. 在“数据包和类保护”窗口中，查找包含敏感 API 项目的数据包的名称。 d. 展开数据包内容。 e. 选择每个需要访问控制的 API 元素。

6. 单击**确定**。
7. 重新编译项目。

使用代码签名密钥保护运行时存储区数据

1. 导入以下类：
   - `java.util.Hashtable`
   - `net.rim.device.api.system.RuntimeStore`
2. 为要存储在运行时存储区中的对象创建哈希 ID。

```
long MY_DATA_ID = 0x33abf322367f9018L;
Hashtable myHashtable = new Hashtable();
```

3. 将对象存储在运行时存储区中，并使用 `CodeSigningKey` 对象保护对象。只有使用密钥签名的应用程序才能读取或更改对象。

```
RuntimeStore.put( MY_DATA_ID, new ControlledAccess( myHashtable, key ) );
```

4. 确保使用特定代码签名密钥保护对象，并调用 `RuntimeStore.get`，其中将对象的哈希 ID 和 `CodeSigningKey` 对象用作参数。

使用代码签名密钥保护持久数据

1. 导入以下类：
   - `java.util.Hashtable`
   - `net.rim.device.api.system.PersistentObject`
2. 为要存储在持久对象中的对象创建哈希 ID。

```
long MY_DATA_ID = 0x33abf322367f9018L;
Hashtable myHashtable = new Hashtable();
```

3. 将对象存储在持久对象中，并使用 `CodeSigningKey` 对象保护对象。例如，在 BlackBerry 设备应用程序运行以下代码行后，只有使用 RSAE .key 文件签名的代码文件才能在持久对象中读取或写入对象。

```
persistentObject.setContents( new ControlledAccess( myHashtable, key ) );
```

4. 确保对象受保护，并调用 `getContents`，其中将 `CodeSigningKey` 对象用作参数。

```
Hashtable myHashtable = (Hashtable) persistentObject.getContents( key );
```

测试 BlackBerry 设备应用程序

5

在 BlackBerry Smartphone Simulator 上测试应用程序

开发和编译应用程序后，可在 BlackBerry® 设备上进行测试。 最常见的第一步是将 BlackBerry® Java® Development Environment 设置为使用 BlackBerry® Smartphone Simulator。 BlackBerry Smartphone Simulator 运行与 BlackBerry 设备相同的 Java 代码，因此 BlackBerry Smartphone Simulator 可提供用于测试应用程序在 BBlackBerry 设备上的运行效果的准确环境。 BlackBerry JDE 包括 BlackBerry Smartphone Simulator 的当前版本。 要下载 BlackBerry Smartphone Simulator 的其他版本，请访问 www.blackberry.com/developers/index.shtml。

在 BlackBerry 设备上测试应用程序

在 BlackBerry® Smartphone Simulator 上测试应用程序后，可将应用程序安装在 BlackBerry 设备上。 如果应用程序使用已签名 API，则可能需要代码签名密钥。 将应用程序安装在 BlackBerry 设备上后，可打开应用程序并测试其功能和性能。

鉴于调试目的，可将设备连接至 BlackBerry® Integrated Development Environment 并使用调试工具逐步执行应用程序代码。 如果尝试确定网络或 Bluetooth 是 Bluetooth SIG 的商标。 问题，或其他难以模拟的问题，BlackBerry IDE 将很有用。

使用已编译 .cod 文件测试应用程序

使用 BlackBerry® Integrated Development Environment 构建项目时，BlackBerry IDE 会将源文件编译为 Java® 字节码，执行预验证，以及为 BlackBerry® 设备应用程序创建单个 .cod 文件和 .jad 文件。

如果 BlackBerry 设备应用程序包含的字节码或资源数据多于 64 KB，BlackBerry IDE 将创建包含同级 .cod 文件的 .cod 文件。 只有 BlackBerry® Browser 才支持包含同级 .cod 文件的 .cod 文件的无线安装。 要确定 .cod 文件是否包含同级 .cod 文件，请提取 .cod 文件的内容。 原始 .cod 文件内的任何 .cod 文件都是同级文件。

要标识 BlackBerry 设备应用程序需要但未随其提供的模块，请检查 Java® 应用程序描述符 (.jad) 文件 RIM-COD-Module-Dependencies 属性。

安装和删除 .cod 文件以测试

要在测试 BlackBerry® 设备应用程序时加载、删除或保存 .cod 文件，请使用 BlackBerry® Java® Development Environment 附带的 JavaLoader 工具。 对于生产应用程序，请使用 BlackBerry® Desktop Software。 您必须按正确顺序加载具有依赖性的 BlackBerry 设备应用程序。 如果项目 A 依赖于项目 B，则先加载项目 B .cod 文件，再加载项目 A .cod 文件。

将 .cod 文件从设备保存至计算机

要在测试 BlackBerry® 设备应用程序时加载、删除或保存 .cod 文件，请使用 BlackBerry® Java® Development Environment 附带的 JavaLoader 工具。

1. 将 BlackBerry 设备连接至计算机。
2. 打开命令提示符,然后导航至 JavaLoader.exe 文件的位置。
3. 请执行以下操作之一：

任务	步骤
将 BlackBerry 设备应用程序 .cod 文件从 BlackBerry 设备保存至计算机。	使用以下格式发出命令：javaloader save *.cod 文件* 例如：javaloader.exe save MyApplication.cod
将同一 .jad 文件中列出的 BlackBerry 设备应用程序 .cod 文件从 BlackBerry 设备保存至计算机。	使用以下格式发出命令：javaloader save *.jad 文件* 例如：javaloader.exe save MyApplication.jad
将同一 CodeModuleGroup 中存储的 BlackBerry 设备应用程序 .cod 文件从 BlackBerry 设备保存至计算机。	使用以下格式发出命令：javaloader save [-g] *模块* 例如：javaloader.exe save -g MyApplication

检索有关 .cod 文件的信息

要在测试 BlackBerry® 设备应用程序时加载、删除或保存 .cod 文件，请使用 BlackBerry® Java® Development Environment 附带的 JavaLoader 工具。

1. 将 BlackBerry® 设备连接至计算机。
2. 打开命令提示符,然后导航至 JavaLoader.exe 文件的位置。
3. 请执行以下操作之一：

任务	步骤
检索 .cod 文件的名称、版本、大小和日期创建信息。	使用以下格式发出命令：`javaloader info` *.cod 文件* 例如：`javaloader.exe info MyApplication.cod`
检索 .cod 文件需要运行的 .cod 文件列表。	使用以下格式发出命令：`javaloader info [-d]` *.cod 文件* 例如：`javaloader.exe info -d MyApplication.cod`
检索以下相关信息 • 同级 .cod 文件 • 代码段的大小 • 数据段的大小 • 已初始化数据的大小 • 类定义的数量 • 应用于 .cod 文件的签名列表	使用以下格式发出命令：`javaloader info [-v]` *.cod 文件* 例如：`javaloader.exe info -v MyApplication.cod`

打包和发布 BlackBerry Java Application

6

预验证 BlackBerry 设备应用程序

要减少加载 BlackBerry 设备应用程序时 BlackBerry® 设备执行的处理量，请部分验证类。 您也可使用 BlackBerry® Smartphone Simulator 预验证 .cod 文件。

在命令提示符处，键入：

```
preverify.exe [-d] output -classpath directory input; directory
```

通过无线网络的应用程序分配

您可以通过无线网络分配应用程序，这将有助于向 BlackBerry®设备用户提供更好的体验，以及简化至很多人员的应用程序分配，因为您不需要计算机应用程序。 BlackBerry 设备用户可通过无线网络安装应用程序。

无法拉（用户启动）

您可以在公共或私有网站上发布已编译应用程序。 BlackBerry® 设备用户可使用 BlackBerry 设备上的浏览器并通过无线网络访问网站，以下载应用程序。 浏览器会提示用户安装应用程序，接着应用程序通过无线网络下载，然后安装在 BlackBerry 设备上。

无线推送（服务器启动）

在 BlackBerry® Enterprise Server 环境中，管理员可通过无线网络将应用程序推送至 BlackBerry 设备用户，以强制安装。 管理员创建新政策并指定 BlackBerry 设备需要应用程序。 应用程序无需任何用户交互即可推送至用户。 将新应用程序发送给大量 BlackBerry 设备用户时，组织会发现此方法很有用。

通过无线网络发布 BlackBerry Java Application

提取同级 .cod 文件。

要确保 BlackBerry® 设备用户不会覆盖原始 .cod 文件，请在内容服务器上，将同级 .cod 文件提取至与原始文件所在的目录不同的目录中。

1. 解压缩原始 .cod 文件并提取同级 .cod 文件。
2. 将每个同级 .cod 文件都放在内容服务器上。

3. 在 .jad 文件中,单独列出同级 .cod 文件。将以下命名约定用于同级 .cod 文件: *原始 .cod 文件的名称 - 序列号*。
4. 为每个同级 .cod 文件创建 RIM-COD-URL-<#> 参数,然后将同级文件的名称放在此参数的右侧。**#** 是以 1 开头并针对每个同级文件增加 1 的编号。将每个同级 .cod 文件的名称都指定为原始 .cod 文件的名称加 **-<#>**。
5. 为每个同级 .cod 文件创建 RIM-COD-Size-<#> 参数,然后将同级文件的大小放在此参数的右侧。**#** 是附加至同级文件的名称的同一编号。将 RIM-COD-Size-<#> 参数放在 RIM-COD=URL-<#> 参数下方。

示例:在 .jad 文件中列出同级 .cod 文件

以下示例在原始 .cod 文件 myAPP 后包含两个名为 **myApp-1.cod** 和 **myApp-2.cod** 的同级文件。 开发人员可将 .cod 文件扩展名附加至每个同级文件名。 开发人员可为每个同级文件创建 RIM-COD-Size-<#> 参数。

```
Manifest-Version: 1.0
MIDlet-Version: 1.0.0
MIDlet-1: ,,
RIM-COD-Module-Dependencies: net_rim_cldc
MicroEdition-Configuration: CLDC-1.0
RIM-COD-Module-Name: MyApp
MIDlet-Name: My Application
RIM-COD-URL: myApp.cod
RIM-COD-Size: 55000
RIM-COD-URL-1: myApp-1.cod
RIM-COD-Size-1: 50000
RIM-COD-URL-2: myApp-2.cod
RIM-COD-Size-2: 25000
MicroEdition-Profile: MIDP-1.0
```

修改 MIDlet 套件的信息

您可以使用 Updatejad 工具(BlackBerry® Java® Development Environment 的一部分)处理 .jad 文件并执行以下操作:

- 更正 .jad 文件中列出的 .cod 文件大小。使用 BlackBerry® Signing Authority Tool 对 .cod 文件进行签名后,.jad 文件中列出的 .cod 文件将更改。
- 创建引用多个 .cod 文件的 .jad 文件。

仅在使用 BlackBerry® Integrated Development Environment 或 RAPC 命令行工具创建且使用 BlackBerry Signing Authority Tool 签名的 .jad 文件上使用 Updatejad 工具。

Updatejad 工具命令具有以下格式:

```
updatejad.exe -q -n input.jad [additional.jad]
```

选项	说明
-q	此选项将抑制 .jad 文件处理操作的成功输出消息的创建。 如果在 .jad 文件处理期间出错，将生成非零退出代码。
-n	此选项抑制原始 .jad 文件的备份。
input.jad	此选项指定要更新的 .jad 文件。
additional.jad	此选项指定要添加至 input.jad 文件的其它属性。

有关详细信息，请参阅 *《BlackBerry Integrated Development Environment Help》* 或 *《BlackBerry Signing Authority Tool version 1.0 - Password Based Administrator Guide》*。

BlackBerry 设备应用程序 .jad 文件的属性

BlackBerry® Integrated Development Environment 让您可以创建两用 .jad 文件，以支持将 MIDlet 下载至 BlackBerry 设备和其它无线设备。 为此，请创建包含 RIM-COD-URL 和 RIM-COD-Size 属性及 MIDlet-Jar-URL 和 MIDlet-Jar-Size 属性的 .jad 文件。 在 BlackBerry 设备上，下载 .cod 文件；在其它无线设备上，下载 .jar 文件。

必需 RIM 属性	说明
RIM-COD-Creation-Time	.cod 文件的创建时间
RIM-COD-Module-Dependencies	.cod 文件需要的模块的列表
RIM-COD-Module-Name	.cod 文件包含的模块的名称
RIM-COD-SHA1	.cod 文件的 SHA1 哈希
RIM-COD-Size	.cod 文件的大小（以字节为单位）
RIM-COD-URL	可从中加载 .cod 文件的 URL

可选 RIM 属性	说明
RIM-Library-Flags	保留用于 Research In Motion
RIM-MIDlet-Flags	保留用于 RIM
RIM-MIDlet-NameResourceBundle	BlackBerry 设备应用程序依赖的资源捆绑包的名称
RIM-MIDlet-Position	BlackBerry 设备应用程序图标在主屏幕上的建议位置可能不是图标在主屏幕上的实际位置

更正 .jad 文件中列出的 .cod 文件大小。

1. 使用 BlackBerry® Integrated Development Environment 创建两个 BlackBerry® 设备应用程序文件，如 **test.cod** 和 **test.jad**。
2. 使用 BlackBerry® Signing Authority Tool 对 .cod 文件进行签名。
3. 从命令提示符处，导航至 Updatejad 工具的位置。
4. 键入命令，以更正 test.jad 中列出的 .cod 文件大小。

```
updatejad.exe test.jad
```

创建引用多个 .cod 文件的 .jad 文件。

1. 使用 BlackBerry® Integrated Development Environment 创建两个 BlackBerry® 设备应用程序文件，如 **lib.cod** 和 **lib.jad**。
2. 使用 BlackBerry® Signing Authority Tool 对 .cod 文件进行签名。
3. 使用 BlackBerry IDE 创建其它两个使用 .jad 文件的 BlackBerry 设备应用程序文件，如 **test.cod** 和 **test.jad**。
4. 使用 BlackBerry Signing Authority Tool 对新 .cod 文件进行签名。
5. 从命令提示符处，导航至 Updatejad 工具的位置。
6. 键入命令，以将 .cod 文件名从第一个 .jad 文件添加至新文件。

```
updatejad.exe test.jad lib.jad
```

使用 BlackBerry Desktop Software 发布 BlackBerry 设备应用程序

BlackBerry 设备应用程序 .alx 文件中的元素

要素	属性	说明
application	id platformVersion blackBerryVersion	application 元素包含单一 BlackBerry® 设备应用程序的元素。

要素	属性	说明
		application 元素还可包含附加嵌套 application 元素。 嵌套让您可以要求，在 BlackBerry 设备上加载 BlackBerry 设备应用程序时，还将在 BlackBerry 设备上加载其必需模块。 id 属性指定 BlackBerry 设备应用程序的唯一标识符。 要提供唯一性，请按相反顺序使用包括公司域的 ID。 例如，com.rim.samples.docs.helloworld。 platformVersion 属性指定 BlackBerry 设备应用程序需要的 BlackBerry 设备上的操作系统软件的版本。 blackBerryVersion 属性指定 BlackBerry 设备应用程序需要的 BlackBerry® Device Software 的版本。
copyright	—	copyright 元素提供显示在 BlackBerry® Desktop Manager 的应用程序加载器工具中的版权信息。
description	—	description 元素提供 BlackBerry 设备应用程序的简短说明。此说明将显示在 BlackBerry Desktop Manager 的应用程序加载器工具中。
directory	platformVersion blackBerryVersion	directory 元素提供文件集的位置。 directory 元素可选。 如果未指定目录，则这些文件必须与 .alx 文件位于相同的位置。 directory 元素指定相对于 .alx 文件位置的目录。 directory 元素在 BlackBerry 设备应用程序内将累计。 例如：

```
<application id="com.abc.my.app">
<directory>MyCodFiles</directory>
<fileset Java="1.0">
<files>
a.cod //resolves to <.alx location>
\MyCodFiles
b.cod
</files>
</fileset>
<directory>MyCodFiles</directory>
```

要素	属性	说明

		`<fileset Java="1.0">`
		`<files>`
		`c.cod //resolves to <.alx location>`
		`\MyCodFiles\MyCodFiles`
		`d.cod`
		`</files>`
		`</fileset>`
		`</application>`
		
		platformVersion 属性指定 BlackBerry 设备应用程序需要的 BlackBerry 设备上的操作系统软件的版本。
		
		blackBerryVersion 属性指定 BlackBerry 设备应用程序需要的 BlackBerry Device Software 的版本。

files	—	files 元素在单个目录中提供要在 BlackBerry 设备上加载的一个或多个 BlackBerry 设备应用程序 .cod 文件的列表。
fileset	Java radio langid Colour platformVersion blackBerryVersion	fileset 元素包括可选 directory 元素和一个或多个 files 元素。 它在单个目录中指定要加载至 BlackBerry 设备上的 .cod 文件集。 要从多个目录加载文件，请在 .alx 文件中包括一个或多个 fileset 元素。 Java 属性指定 .cod 文件与其兼容的 BlackBerry® Java® Virtual Machine 的最低版本。 Java 属性为必需。 radio 属性让您可在 BlackBerry 设备上加载各种 BlackBerry 设备应用程序或模块，具体取决于 BlackBerry 设备的网络类型。 可能值包括 Mobitex□DataTAC□GPRS□CDMA 和 IDEN。 radio 属性可选。 langid 属性让您可加载各种 BlackBerry 设备应用程序或模块，具体取决于 BlackBerry 设备用户添加至 BlackBerry 设备的语言支持。 此值是 Win32 langid 代码；例如：0x0009（英语）、0x0007（德语）、0x000a（西班牙语）和 0x000c（法语）。 langid 属性可选。

要素	属性	说明
		colour 属性让您可加载各种要彩色或单色显示的 BlackBerry 设备应用程序或模块。 此值为 Boolean；true 表示彩色显示，而 false 表示单色。 platformVersion 属性指定 BlackBerry 设备应用程序需要的 BlackBerry 设备上的操作系统软件的版本。 blackBerryVersion 属性指定 BlackBerry 设备应用程序需要的 BlackBerry Device Software 的版本。
hidden	—	hidden 元素隐藏数据包，以便不会在应用程序加载器中将其显示给 BlackBerry 设备用户。 要隐藏数据包，请添加以下行：<hidden>true</hidden>。 默认情况下，将此元素与 required 元素一起使用，以加载 BlackBerry 设备应用程序；如果存在其它 BlackBerry 设备应用程序，则设置 requires 标记以加载此数据包。 只有公司系统管理员才能使用 hidden 标记。 第三方软件供应商无法使用此标记。 BlackBerry Desktop Software 3.6 版或更高版本支持此元素。
language	langid	应用程序加载器以 langid 属性指定的语言运行时，language 元素让您可以覆盖应用程序加载器中显示的文本。 要支持多种语言，请指定多个 language 元素。 要指定每种语言的 name、description、version、vendor 和 copyright 元素，请在 language 元素中嵌套这些元素。 如果未嵌套元素，则文本将以默认语言显示。 langid 属性指定此信息适用的语言的 Win32 langid 代码。 例如，某些 Win32 langid 代码包括：0x0009（英语）、0x0007（德语）、0x000a（西班牙语）和 0x000c（法语）。

要素	属性	说明
library	id	您可以使用 library 元素替代 application 元素。它包含单个库模块的元素。 您无法嵌套模块。 默认情况下，库模块不会显示在 BlackBerry Desktop Manager 的应用程序加载器工具中。 通常将 library 元素用作 <requires> 元素的目标，以便在将特定 BlackBerry 设备应用程序加载至 BlackBerry 设备上时，必需库也将加载至 BlackBerry 设备上。 BlackBerry Desktop Software 3.6 版或更高版本支持此元素。
loader	version	loader 元素包含一个或多个 application 元素。 version 属性指定 BlackBerry Desktop Manager 的应用程序加载器工具的版本。
name	—	name 元素提供 BlackBerry 设备应用程序的描述性名称。此名称将显示在 BlackBerry Desktop Manager 的应用程序加载器工具中。
required	—	required元素让您可强制加载应用程序。 BlackBerry Desktop Manager 的应用程序加载器工具将选择要安装的 BlackBerry 设备应用程序，而 BlackBerry 设备用户无法更改此类选择。 添加以下行：<required>true</required>。 只有公司系统管理员才能使用 required 标记。 第三方软件供应商无法使用此标记。 BlackBerry Desktop Software 3.5 版或更高版本支持此元素。
requires	id	requires 元素是指定此 BlackBerry 设备应用程序依赖的数据包的 id的可选元素。 如果 BlackBerry 设备应用程序依赖多个 BlackBerry 设备应用程序，此元素可出现多次。 在将 BlackBerry 设备应用程序加载至 BlackBerry 设备上时，<requires> 标记指定的所有数据包也将加载至 BlackBerry 设备上。

要素	属性	说明
		BlackBerry Desktop Software 3.6 版或更高版本支持此元素。
vendor	—	vendor 元素提供创建 BlackBerry 设备应用程序的公司的名称。此说明将显示在 BlackBerry Desktop Manager 的应用程序加载器工具中。
version	—	version 元素提供 BlackBerry 设备应用程序的版本号。此版本号将显示在 BlackBerry Desktop Manager 的应用程序加载器工具中。 此版本号仅供参考。

通过计算机连接的应用程序分配

从一台计算机分配应用程序

您可以使用 BlackBerry® Desktop Manager 中的应用程序加载器工具将应用程序安装在 BlackBerry 设备上。 应用程序加载器工具可向用户提供将应用程序从计算机下载至 BlackBerry 设备的简单方式。

从网页分配应用程序

您可以使用 BlackBerry® Application Web Loader 在网站上发布已编译应用程序。 用户可以在计算机上使用 Windows® Internet Explorer®，以访问网页并将应用程序安装在 BlackBerry 设备上。 在 BlackBerry 设备用户访问网页时，BlackBerry Application Web Loader 会提示将设备连接至 USB 端口。 他们接着可以使用 ActiveX® 控件安装应用程序。 BlackBerry Application Web Loader 可向 BlackBerry 设备用户提供无需运行 BlackBerry® Desktop Manager 就从计算机安装应用程序的简单方式。

分配要测试的应用程序

BlackBerry® Java® Development Environment 包括名为 JavaLoader 工具的命令行工具，此工具位于 BlackBerry JDE 文件夹的 BIN 文件夹中。 您可以使用 JavaLoader 工具直接通过 USB 端口在 BlackBerry 设备上快速安装和删除已编译应用程序文件。 您无需任何描述符文件或网页。 测试和开发期间频繁安装和删除应用程序时，JavaLoader 工具可能很有用；但是，JavaLoader 工具不是设计为由 BlackBerry 设备用户使用。

从计算机发布应用程序

创建应用程序加载器文件

通过使用 BlackBerry® Desktop Manager 的应用程序加载器工具，可使用应用程序加载器文件发布 BlackBerry® 设备应用程序。

1. 为每个 BlackBerry 设备应用程序创建 .alx 文件，然后使用 .cod 文件将 .alx 文件分配给 BlackBerry 设备用户。有关设备的逐步说明，请参阅设备随附的 *应用程序加载器联机帮助* 以了解 .alx 文件的详细信息。
2. 在 BlackBerry® Integrated Development Environment 中，选择项目。
3. 在**项目**菜单中，单击**生成 .alx 文件**。

将 BlackBerry 设备应用程序安装在特定设备上

1. 打开文本编辑器。
2. 找到 BlackBerry® 设备应用程序的 .alx 文件。
3. 在 .alx 文件中，确保 fileset 开始标记中的 series 属性引用要在其上安装 BlackBerry 设备应用程序的 BlackBerry 设备。

   ```
   <fileset series="8700" Java="1.0">
   ```

   有关 series 属性的详细信息，请参阅位于 BlackBerry® Java® Development Environment 安装目录的模拟器目录中的 Platform.alx 文件：

   Program Files\Research In Motion\BlackBerry JDE 4.6.0\simulator。
4. 确保 files 标记包含 BlackBerry 设备应用程序的 .cod 文件的引用。

   ```
   <files>
   My_application.cod
   </files>
   ```
5. 更新 application、description 和其它标记，以反映 .alx 文件的用途。

   ```
   <application id="Push only to 8700">
   ...
   <description>This will push the COD only to 8700s</description>
   ```

代码示例：在特定 BlackBerry 设备上加载 BlackBerry 设备应用程序

```
<loader version="1.0">
<application id="Push only to 8700">
<name>Alien</name>
```

```
<description>This will push the COD only to 8700s</description>
<version>2006.02.14.1838</version>
<vendor>RIM</vendor>
<copyright>Copyright (c) 2001-2005</copyright>
<fileset series="8700" Java="1.0">
<files>
My_application.cod
</files>
</fileset>
</application>
</loader>
```

指定 BlackBerry Device Software 的受支持版本

使用只有特定 BlackBerry® Device Software 版本才提供的 API 的 BlackBerry® 设备应用程序应该使用 `_blackberryVersion` 属性指定支持的 BlackBerry 设备版本。

用于运行在特定平台版本上的 BlackBerry 设备的 BlackBerry 设备应用程序应该使用 `_platformVersion` 属性指定支持的平台版本。 `_platformVersion` 属性可在 directory 标记、application 标记或 fileset 标记内使用。

您可以使用以下规则指定 BlackBerry Device Software 或平台版本的版本范围：

- 中括号 [] 表示包括(闭合)范围匹配。
- 小括号 () 表示排他(开放)范围匹配。
- 缺少范围上限表示 0。
- 缺少范围下限表示无穷大。

例如，[4.0,) 表示介于 4.0 和无穷大之间的任何版本。

代码示例：防止在低于 4.0 版的 BlackBerry Device Software 上加载模块

```
<application id="application_id" _blackberryVersion="[4.0,)">
...
</application>
```

代码示例：为各种 BlackBerry Device Software 版本提供备选模块。

```
<application id="application_id>">
...
<fileset _blackberryVersion="(,4.0)">
... modules for BlackBerry device software versions earlier than 4.0
</fileset>
<fileset _blackberryVersion="[4.0,)">
... modules for BlackBerry device software versions 4.0 and later
</fileset>
</application>
```

代码示例：防止在低于 2.4.0.66 版的平台上加载模块。

```
<application id="application_id" _platformVersion="[2.4.0.66,)">
...
</application>
```

代码示例：为各种平台版本提供备选模块。

```
<application id="application_id>">
...
<fileset _platformVersion="(,2.4.0.66)">
... modules for BlackBerry OS plaform versions earlier than 2.4.0.66
</fileset>
<fileset _platformVersion="[2.4.0.66,)">
... modules for BlackBerry OS plaform versions 2.4.0.66 and later
</fileset>
</application>
```

指定应用程序在 BlackBerry 设备上的位置

您可以将 BlackBerry® 设备应用程序添加至设备上通过主题提供的应用程序文件夹之一，添加至创建的新应用程序文件夹，或添加至设备的主屏幕。 例如，如果开发游戏，则可将游戏添加至 BlackBerry 设备上的“游戏”文件夹。 如果应用程序具有多个入口点，则可为每个入口点指定一个应用程序文件夹。 如果未指定位置，BlackBerry 设备会将应用程序添加至默认应用程序文件夹（例如，“下载”文件夹）。 您可指定安装在运行 BlackBerry® Device Software 5.0 版或更高版本的 BlackBerry 设备上的应用程序的位置。

运行 BlackBerry Device Software 5.0 版的设备上的 Precision 主题提供以下应用程序文件夹。

- 应用程序
- 下载
- 游戏
- 即时消息
- 邮件
- 多媒体
- 音乐
- 设置

如果创建新应用程序文件夹，文件夹名称必须使用有效字符。 例如，不能使用小于号 (<) 或大于号 (>) 等字符。 有关无效字符列表，请参阅 BlackBerry® Java® Development Environment 的 API 参考中的 `FilenameTextFilter` 类的文档。

用户可更改应用程序在设备上的位置。 如果用户更改应用程序的位置，设备始终使用新位置。 例如，如果用户安装、删除并在后面重新安装应用程序，设备会将应用程序添加至用户以前指定的位置。

指定应用程序在 BlackBerry 设备上的位置

通过将 Content-Folder 属性添加至应用程序的 .jad 文件，可在 BlackBerry 设备上指定要将应用程序添加至的位置。 如果指定了应用程序文件夹，但无法创建该文件夹，应用程序将会添加至默认文件夹（例如，“下载”文件夹）。

开始之前：确定您的 BlackBerry® 设备运行 BlackBerry® Device Software 5.0 或更高版本。

1. 在 Eclipse® 中，打开并打包您的 BlackBerry 设备应用程序项目。
2. 在 Package Explorer 视图中，打开 *<project_folder>*/deliverables/Standard/*<x.x.x>* 文件夹中的 .jad 文件。如果看不到文件夹或 .jad 文件，请确定已将项目打包。
3. 在文本编辑器中，在 .jad 文件末尾处新添加一行，并键入以下内容：
   - 要将应用程序添加至设备的主屏幕，请键入 **Content-Folder: /**。
   - 要将应用程序添加至应用程序文件夹，请键入 **Content-Folder: *<folder_name>***。
   - 要指定多个入口点的位置，请为每个入口点键入 **Content-Folder-*x*: *<folder_name>***。例如，为第一个入口点键入 Content-Folder-1:Games，为第二个入口点键入 Content-Folder-2: Applications。
4. 保存 .jad 文件。
5. 在 Eclipse 的 Package Explorer 视图中，将 .jad 文件从 *<project_folder>*/deliverables/Standard/*<x.x.x>* 文件夹拖放到您应用程序项目的根文件夹中。
6. 将项目打包。
7. 使用新数据包文件测试和发布应用程序。

示例：通过更改 .jad 文件将应用程序添加至“游戏”文件夹

```
Manifest-Version: 1.0
MIDlet-Version: 1.0.0
MIDlet-Jar-Size: 2912
MicroEdition-Configuration: CLDC-1.1
MIDlet-Jar-URL: MyGame.jar
RIM-COD-Module-Dependencies: net_rim_cldc
RIM-MIDlet-Flags-1: 0
RIM-COD-Module-Name: MyGame
MIDlet-Name: MyGame
RIM-COD-Size: 1504
RIM-COD-Creation-Time: 1272372497
MIDlet-1: My Game,img/mygame.png,
RIM-COD-URL: MyGame.cod
RIM-COD-SHA1: 82 2f 22 b6 e6 34 ef c3 2b 0e a4 96 22 08 c0 60 39 4d db aa
MicroEdition-Profile: MIDP-2.0
MIDlet-Vendor: Research In Motion Ltd.
Content-Folder: Games
```

将 BlackBerry 设备应用程序本地化 7

多语言支持

BlackBerry® Integrated Development Environment 包括用于创建字符串资源的资源机制。 本地化 API 是 net.rim.device.api.i18n 数据包的一部分。 MIDP 应用程序不支持本地化。

BlackBerry Integrated Development Environment 将区域设置的资源存储在 ResourceBundle 对象中。ResourceBundleFamily 对象包含 ResourceBundle 集合，此集合对应用程序的资源进行分组。 应用程序可切换语言，无需新资源捆绑包，具体取决于 BlackBerry 设备用户的区域设置。

您可以使用 BlackBerry Integrated Development Environment 将每个资源捆绑包都编译为独立编译的 .cod 文件。 您可以使用应用程序的相应 .cod 文件将其他 .cod 文件加载至 BlackBerry 设备。

资源根据继承按层次结构组织。 如果未在区域设置中定义字符串，则使用下个最近区域设置中的字符串。

需要本地化的文件

需要本地化的文件	说明	示例
资源题头文件	此文件定义每个本地化字符串的描述性密钥。 在 BlackBerry® Integrated Development Environment 构建项目时，将创建资源接口，其中 Resource 将附加至 .rrh 文件名。 例如，如果创建 AppName.rrh，则接口的名称将为 AppNameResource。	AppName.rrh
资源内容文件（根区域设置）	此文件将资源密钥映射至根（全局）区域设置的字符串值。 其名称与资源题头文件名相同。	AppName.rrc
资源内容文件（特定区域设置）	此文件将资源密钥映射至特定区域设置（语言和国家（地区））的字符串值。 文件名为资源题头文件名加下划线（_）和语言代码，再加（可选）下划线（_）和国家（地区）代码。 将资源内容文件保存至 .java 文件所在的文件夹。 例如，在包含 CountryInfo.java 的文件夹中，保存 save CountryInfo.rrc（根区域设置）、CountryInfo_en.rrc（英语）和 CountryInfo_fr.rrc（法语）。	AppName_en.rrc AppName_en_GB.rrc AppName_fr.rrc

需要本地化的文件	说明	示例
初始化文件	此文件将初始化资源捆绑包机制。 只有将资源作为独立项目编译，才需要此文件。	init.java

管理 BlackBerry 设备应用程序套件的本地化文件

如果创建 BlackBerry® 设备应用程序套件，请将资源整理至每个区域设置的独立项目中。 BlackBerry® Integrated Development Environment 提供内置初始化机制。 您仅需要使用空 main() 创建空初始化类。 如果支持大量区域设置，请为所有资源题头（.rrh）文件创建单个库对象，并将项目类型设置为**库**。对于库中的每个资源区域设置，都定义项目之间的依赖性。

1. 打开 BlackBerry® Integrated Development Environment。
2. 为每个资源捆绑包(区域设置)创建项目,包括根区域设置。
3. 将每个资源区域设置的项目的名称都指定为根区域设置的项目名称,加双下划线 (__)、语言代码和(可选)下划线 (_) 加国家(地区)代码。例如,如果根区域设置项目的名称为 **com_company_app**,则每个区域设置的项目的名称将为 **com_company_app__en**、**com_company_app__en_GB** 和 **com_company_app__fr**。
4. 右键单击项目,然后单击**属性**。
5. 在**构建**选项卡的**输出文件名**字段中,键入已编译文件的名称,其中不包括文件名扩展名。
6. 创建初始化文件。

```
package com.rim.samples.device.resource;
import net.rim.device.api.i18n.*;
public class init {
public static void main (String[] args) { }
}
```

7. 为每个 BlackBerry 设备应用程序创建一个资源题头文件。
8. 将资源题头（.rrh）文件复制到每个 BlackBerry 设备应用程序的项目。
9. 将资源题头文件复制到每个资源区域设置项目。
10. 为每个 BlackBerry 设备应用程序创建一个资源内容文件。
11. 为每个支持的区域设置创建一个资源内容文件。
12. 在每个资源区域设置项目中,右键单击每个 .rrh 文件,然后单击**属性**。
13. 选择**仅限依赖性。不构建**。
14. 将资源内容（.rrc）文件添加至相应区域设置的项目。

自定义用户身份验证

8

要登录 BlackBerry 设备，用户必须生成安全令牌。 默认情况下，BlackBerry 设备用户将键入密码，以验证身份并生成对应的安全令牌。 运行 BlackBerry® Device Software 5.0 或更高版本的 BlackBerry® 设备还支持自定义用户身份验证模块。

您必须创建用户身份验证模块，才能将附加硬件（如智能卡或生物特征识别设备）用于身份验证。BlackBerry 设备上的安全框架使用用户身份验证模块与硬件交互。

有关自定义用户身份验证和 `net.rim.device.api.userauthenticator` 数据包的详细信息，请参阅 BlackBerry® Java® Development Environment 的 API 参考。

词汇表

9

3GPP
Third Generation Partnership Project（第三代合作伙伴项目）

AES
Advanced Encryption Standard（高级加密标准）

API
Application Programming Interface（应用程序编程接口）

APN
Access Point Name（访问点名称）

ASCII
美国国家标准学会

.alx 文件
.alx 文件是应用程序描述器，向 BlackBerry 设备提供有关 BlackBerry Java® Application 及该应用程序的 .cod 文件位置的信息。

BlackBerry MDS
BlackBerry® Mobile Data System（BlackBerry® 移动数据系统）

CDMA
Code Division Multiple Access（码分多址）

COM 端口
通信端口

EDGE
Enhanced Data Rates for Global Evolution（全球发展的增强数据速率）

EVDO
Evolution Data Optimized（演进数据最优化）

GAN
Generic Access Network（通用接入网络）

GERAN
GSM-EDGE 无线接入网络

GPRS

General Packet Radio Service（通用分组无线业务）

GSM
Global System for Mobile communications®（全球移动通信系统）

HTTP
Hypertext Transfer Protocol（超文本传输协议）

HTTPS
Hypertext Transfer Protocol over Secure Sockets Layer（基于安全套接字层的超文本传输协议）

IPPP
Internet 协议代理协议

JSR
Java® Specification Request（Java® 规范请求）

MIDP
Mobile Information Device Profile（移动信息设备配置文件）

PAP
推入访问协议

PIN
Personal identification number（个人标识号）

RAPC
RIM 应用程序编译器

服务预订
服务预订确定 BlackBerry 设备或采用 BlackBerry 技术的设备中可供使用的服务。

SSID
Service Set Identifier（服务集标识符）

TCP
Transmission Control Protocol（传输控制协议）

TLS
传输层安全协议

Triple DES
三重数据加密标准

UDP

User Datagram Protocol（用户数据报协议）

UMTS
Universal Mobile Telecommunications System（通用移动通信系统）

UTRAN
UMTS 陆地无线接入网络

WAP
Wireless Application Protocol（无线应用协议）

WLAN
无线局域网

提供反馈

10

要提供关于此交付项目的反馈，请访问 www.blackberry.com/docsfeedback。

相关资源 11

有关为 BlackBerry® 设备开发应用程序的详细信息，请参阅以下资源。

- 《BlackBerry Java Application Fundamentals Guide》
- 《BlackBerry Java Application 用户界面和导航开发指南》
- 《BlackBerry Java Application Integration Development Guide》
- 《BlackBerry Java Application 多媒体开发指南》
- 《BlackBerry Java Application Accessibility Development Guide》
- 《BlackBerry Java Application Transitioning to Touch Screen Development Technical Note》
- 《BlackBerry Browser Development Guide》
- BlackBerry Developer Zone 位于 www.blackberry.com/developers
- BlackBerry® Java® Development Environment 的 API 参考

文档修订历史记录 12

日期	说明
2010 年 4 月 27 日	已更改 指定应用程序在 BlackBerry 设备上的位置 主题。
2010 年 4 月 6 日	已添加以下主题： • 自定义用户身份验证 已更新以下主题： • 网络连接和传输类型 • 文件系统和路径 已添加以下代码示例： • 代码示例：读取二进制文件的各个部分
2009 年 10 月 8 日	已添加以下主题： • 指定应用程序在 BlackBerry 设备上的位置 • 指定应用程序在 BlackBerry 设备上的位置
2009 年 8 月 20 日	已添加显示如何使用网络 API 的功能的主题和代码示例。 已添加文件系统和路径引用信息。
2009 年 8 月 14 日	已更改系统要求主题。

法律声明 13

本文档包含的限制、排除事项和免责声明应适用于：(A) 不考虑操作、需求或用户操作的原因性质，包括但不限于违约、疏忽、侵权行为、严格赔偿责任或任何其他法律理论且应克服根本性违约、违约、此协议基本目的失败、或内含的任何补救措施；和 (B) RIM 及其附属公司，其继任人、分配、代理、供应商（包括开播服务提供商）、授权 RIM 分销商（也包括开播服务提供商）及其董事、雇员和独立承包商。

除上述限制和排除事项外，RIM 及其附属公司的任何董事、雇员、代理、分销商、供应商、独立承包商对由本文档引起或相关的事故概不承担任何责任。

在订购、安装或使用任何第三方产品和服务前，用户有责任确保其开播服务提供商已同意支持所有功能。某些无线服务提供商可能不会在订购 BlackBerry® Internet Service 时提供 Internet 浏览功能。 请与服务提供商联系，以了解可用性、漫游安排计划、服务计划和功能。 安装或使用具有 RIM 产品和服务的第三方产品和服务可能会要求一个或多个专利、商标、版权或其他许可证以避免侵害或违反第三方权利。 您应独自负责确定是否使用第三方产品和服务，如果任何第三方许可证要求如此。 如果有此要求，则您有责任获取这些许可证。 除非已获取所有必需的许可证，否则您不应安装或使用第三方产品和服务。 对于为了方便而随 RIM 产品和服务一起提供的和按“原样”形式（不具有 RIM 所做的任何种类的明示或暗示条件、认可、保证、陈述或担保）提供的任何第三方产品和服务，RIM 概不承担任何责任。 除了许可证已清楚表明或与 RIM 签订的其他协议，您使用第三方产品和服务应该受您同意这些产品或服务的单独许可证和其他第三方适用协议条款所约束。

本说明文档中介绍的某些功能可能需要安装最低版本的 BlackBerry® Enterprise Server、BlackBerry® Desktop Software 和/或 BlackBerry® Device Software。

此外已在单独的许可证或 RIM 适用的其他协议中陈述了使用任何 RIM 产品或服务的条款。 对于除本文档之外任何部分的 RIM 产品或服务，本文档中的任何内容不得用于代替由 RIM 提供的任何明确书面协议或担保。

Research In Motion Limited
295 Phillip Street
Waterloo, ON N2L 3W8
Canada

Research In Motion UK Limited
Centrum House
36 Station Road
Egham, Surrey TW20 9LF
United Kingdom

加拿大出版